

Aplicació del *Material Point Method (MPM)* a la deformació d'objectes

Introducció a la simulació de materials
deformables

Martín Garcia, Pol

Treball de final de grau d'Enginyeria Informàtica

Especialitat de Computació
Defensa 23 de Gener, 2020

Director: Susín Sánchez, Toni
Dpt. MAT

Resum

Una de les branques més importants dins de l'animació per computador és la simulació física del comportament dels objectes d'una escena. Generalment es busca obtenir resultats que visualment es comportin com els objectes reals, i per això és important utilitzar els models matemàtics que defineixen el seu comportament.

El fet que la validació en molts casos sigui visual, permet poder dur a terme simplificacions en els models matemàtics per guanyar velocitat de càlcul o disminuir la complexitat del que seria una simulació d'enginyeria. Això si, ajustant sempre les possibles simplificacions a un comportament final visualment realista.

Els objectes amb més dificultat a l'hora de la simulació són els objectes deformables, el comportament dels quals ve definit per la mecànica dels medis continus. En aquesta categoria podem trobar des de fluids fins a diferents tipus de materials elàstics. Així podrem simular des d'aigua fins a una pilota de goma, passant per sorra o objectes quasi rígids que es deformen de manera permanent (deformació plàstica).

El canvi dels materials vindrà definit pels paràmetres que farem servir en les simulacions, com ara densitat, viscositat, elasticitat, etc. Per tant ens interessa fer servir un model de simulació que permeti canviar i experimentar amb tots aquests valors.

En aquest treball s'aprofundeix en el mètode *MPM* (*material point method*) que és un mètode híbrid basat en partícules i malles, prou versàtil i eficient per a realitzar una àmplia gamma de simulacions amb diversos materials. Amb aquest mètode podrem incorporar també els diferents tipus de deformacions (elàstica, plàstica i de trencament) que poden experimentar els objectes simulats. També, aquest treball intenta ser una bona introducció en el camp de la simulació de materials des del punt de vista d'un enginyer informàtic, sense assumir coneixements previs de mecànica dels medis continus, mètodes numèrics, i tècniques de simulació, tan basades en elements finits (*FEM*) com en el mateix *MPM*.

Per això, aquest document es redacta intentant explicar els múltiples conceptes necessaris per entendre el funcionament dels simuladors actuals. S'han inclòs capítols i apèndixs, així com les referències a la corresponent literatura pròpia d'aquest camp.

Finalment s'ha desenvolupat un simulador 3D de materials deformables seguint el model de *MPM-MLS*, juntament amb una interfície interactiva que permet definir una escena amb els respectius materials per fer-ne la simulació, i posterior representació gràfica, emmagatzematge i exportació.

Resumen

Una de las ramas más importantes dentro de la animación por computador es la simulación física del comportamiento de los objetos en una escena. Generalmente se busca obtener resultados que visualmente se comporten como objetos reales, y por tanto es importante utilizar los modelos matemáticos que definen su comportamiento.

El hecho que la validación en muchos casos sea visual, permite llevar a cabo simplificaciones en los modelos matemáticos para ganar velocidad de cálculo o disminuir la complejidad de la que sería una simulación de ingeniería. Eso si, ajustando siempre las posibles simplificaciones a un comportamiento final visualmente realista.

Los objetos con más dificultad en el momento de la simulación son los objetos deformables, el comportamiento de los cuales se define por la mecánica de medios continuos. En esta categoría podemos encontrar desde fluidos hasta diferentes tipos de materiales elásticos. Así podremos simular desde agua hasta una pelota de goma, pasando por arena u objetos casi rígidos que se deforman permanentemente (deformación plástica).

El cambio de los materiales vendrá definido por los parámetros que usemos en las simulaciones, como la densidad, viscosidad, elasticidad, etc. Por tanto nos interesa usar un modelo de simulación que permita cambiar y experimentar con todos estos valores.

En este trabajo se profundiza en el método *MPM* (*material point method*) que es un método híbrido basado en partículas y mallas, lo suficientemente versátil y eficiente para realizar una amplia gama de simulaciones con diversos materiales. Como este método podremos incorporar también los diferentes tipos de deformaciones (elástica, plástica y de rotura).

También, este trabajo intenta ser una buena introducción en el campo de la simulación de materiales desde el punto de vista de un ingeniero informático, sin asumir conocimientos previos de mecánica de medios continuos, métodos numéricos, y técnicas de simulación, tanto basadas en elementos finitos (*FEM*) como en el mismo *MPM*.

Por eso, este documento se redacta intentando explicar los múltiples conceptos necesarios para entender el funcionamiento de los simuladores actuales. Se han incluido capítulos y apéndices, así como las referencias a la correspondiente literatura propia de este campo.

Finalmente se ha desarrollado un simulador 3D de materiales deformables siguiendo el modelo de *MPM-MLS*, juntamente con una interfaz interactiva que permite definir una escena con los respectivos materiales para hacer la simulación, y posterior representación gráfica, almacenaje y exportación.

Abstract

One of the most important branches of computer animation is the physic simulation of objects' behavior in a scene. It is usually wanted to obtain results that visually behave as real objects, and thus it is important to use the mathematical models that define their behavior.

The fact that the validation in many cases is visual, allows to simplify such mathematical models to gain calculation speed or to decrease the complexity of what would be an engineering simulation. However, always fitting the possible simplifications to a final behavior visually realistic.

The objects with higher difficulty when it comes to simulate them are deformable objects, the behavior of which is defined by continuum mechanics. In this category we can find from fluids to different types of elastic materials. With this we will be able to simulate from water to a rubber ball, through sand or objects almost rigid that get permanently deformed (plastic deformation).

The change in materials is given by the parameters that will be used in the simulations, such as density, viscosity, elasticity, etc. Because of this we are interested in using a simulation model that allows to change and experiment with all this values.

In this document we delve into *MPM* (material point method) which is a hybrid method based in particles and meshes, adaptable enough and efficient to conduct a wide spectrum of simulations with multiple materials. With this method we can incorporate the different types of deformations (elastic, plastic and tear) that the simulated objects can experience.

Also, this document tries to be a good introduction to the material simulation field from the point of view of a computer scientist, without assuming previous knowledge of continuum mechanics, numerical methods, and simulation techniques, either based on finite element methods (*FEM*) or in *MPM* itself.

Thereby, this text is written in order to explain the multiple necessary concepts to understand the workings of actual simulators. Chapters, appendixes, and references to the corresponding literature of the field are included.

Finally a 3D simulator for deformable materials has been developed following the *MPM-MLS* model, altogether with an interactive interface that allows the definition of a scene with its own materials to simulate, and succeeding graphical representation, storage and exportation.

Índex

1	Introducció	9
1.1	Context	9
1.1.1	Conceptes bàsics	10
1.1.2	Formulació del problema	11
1.2	Estat de l'art	12
1.3	Estructura del document	12
1.4	Sobre la notació	13
2	Deformacions	16
2.1	Cinemàtica	16
2.1.1	Descripció del moviment continu	16
2.1.2	Descripció del material	17
2.1.3	Gradient de deformació	18
2.2	Derivada del gradient de deformació	19
2.3	Tensors de deformació	19
2.4	Tensió	20
2.4.1	Tensió de <i>Cauchy</i>	20
2.4.2	Tracció superficial	21
2.4.3	Tensors de tensió de <i>Piola-Kirchhoff</i>	23
2.5	Hiperelasticitat	26
2.5.1	Funcions energètiques de densitat d'esforç	26
2.5.2	<i>Neo-Hookean</i>	26
2.5.3	Corrotacional Fix	28
2.5.4	Plasticitat	29
3	Necessitats del simulador	31
3.1	Discretització	31
3.1.1	Espai <i>Eulerià</i>	31
3.1.2	Partícules <i>Lagrangianes</i>	32
3.1.3	Temps	32
3.2	Interacció partícula-graella	33
3.2.1	Interpolació	33
3.2.2	<i>PIC</i>	34
3.3	Evolució del gradient de deformació	35

4	<i>Material point method</i>	37
4.1	Mètode complet	37
4.2	<i>APIC</i>	38
4.3	<i>MLS</i>	39
4.3.1	Actualització del gradient de deformació	39
4.3.2	Actualització de la graella	40
4.3.3	Integració explícita de forces	40
4.3.4	Partícula a graella	41
4.4	Plasticitat	41
4.5	Col·lisions externes en graella	43
5	Implementació <i>MPM</i>	44
5.1	Preàmbul	44
5.1.1	Dependències	45
5.1.2	Estructures de dades	45
5.2	Algorisme	46
5.2.1	Inicialització	47
5.2.2	<i>P2G</i>	47
5.2.3	Processament de la graella	51
5.2.4	<i>G2P</i>	53
5.3	Optimitzacions	57
5.3.1	Paral·lelisme	57
5.3.2	Optimització <i>P2G</i>	57
5.3.3	Desenroscat de bucles	60
5.3.4	Graella paginada esparsa	60
5.4	Materials	60
5.5	Complexitat	61
6	Interfícies	62
6.1	Entrada de dades	62
6.1.1	Models de simulació	63
6.1.2	Models de físiques	63
6.2	Emmagatzematge	64
6.3	Visualitzador	64
6.3.1	Renderització	65
6.3.2	Fotogrames	67
6.3.3	Interacció	68
6.4	Exportació	68

7	Resultats	70
7.1	Temps	70
7.2	Resultats de les simulacions	72
7.2.1	Models potencials elàstics	72
7.2.2	Deformació d'objectes	73
7.2.3	Trencament	74
7.2.4	Elasticitat	76
7.2.5	Enduriment	76
7.2.6	Físiques	77
7.2.7	Líquids	78
7.2.8	Acoblament	78
8	Gestió del projecte	80
8.1	Descripció de tasques	80
8.2	Gestió econòmica del projecte	82
8.2.1	Partides	82
8.2.2	Pressupost final	86
8.2.3	Viabilitat econòmica	86
8.3	Informe de Sostenibilitat	86
8.3.1	Dimensions de la sostenibilitat	87
8.3.2	Matriu de sostenibilitat	89
9	Conclusions	90
9.1	Competències tècniques	91
9.2	Treball futur	92
9.3	Agraïments	93
	Índex de figures	94
	Índex de taules	95
	Referències	96
APÈNDIX A.	Teoria	99
A.1	Càlcul vectorial	99
A.2	Equació de Navier-Stokes	102
A.3	Descomposició polar	106
A.4	Descomposició en valors singulars	108
A.5	Derivada del material	108

A.6 Esforç (Strain)	111
A.7 Segon tensor de <i>Piola-Kirchhoff</i>	114
A.8 Equacions de govern	115
APÈNDIX B. Simulacions disponibles	118
APÈNDIX C. Extractes de codi	134
C.1 Generar icosaedre	134
C.2 Parts del simulador	137
C.2.1 Matriu aff \mathbf{Q}	137
C.2.2 Processament de la graella	138
C.2.3 Transferència $G2P$	139
C.3 Producte exterior de tensors 3D	139

1. Introducció

Aquest treball de fi de grau ha estat centrat en la simulació de materials deformables, una temàtica molt àmplia que tot i que conceptualment sembla trivial, és molt complexa. Molts detalls i decisions s'han de tenir en compte a l'hora d'implementar un simulador d'aquestes característiques, com la representació del material, si aquest és compressible, elàstic, interactiu amb col·lisions, etc.

Per a poder presentar el treball correctament, aquest document també serveix com a introducció a la simulació de fluids sense assumir coneixements previs en aquest aspecte, i des del punt de vista d'un enginyer informàtic.

1.1. Context

La simulació de fluids, o dinàmiques de fluids computacionals (*CFD's*), és una disciplina que, mitjançant tècniques d'anàlisi numèrica, busca resoldre problemes que impliquen d'alguna manera un tipus de fluctuació o interacció amb fluids.

La base de qualsevol simulador de fluids són les equacions de Navier-Stokes (explicació i derivació consultable a [A.2.2](#)), que descriuen la mecànica d'aquests; però es requereix molt més que les equacions per a poder implementar un simulador, doncs també són necessaris coneixements de software i hardware per a una implementació eficient, tècniques de gràfics per computador per a una adequada visualització, i coneixements matemàtics d'àlgebra lineal. Tot i això, l'abast del projecte ens obliga a sortir de l'àmbit de coneixements d'un enginyer informàtic, i integrar múltiples conceptes matemàtics de diferents caires, que seran explicats degudament.

Amb l'objectiu de poder sortejar aquest inconvenient, doncs els coneixements que he mencionat anteriorment a la FIB s'instrueixen en assignatures de màster impartides per professors d'altres facultats, el projecte s'està duent a terme sota la supervisió d'un professor del departament de matemàtiques de la UPC, cobrint d'aquesta manera les mancances de coneixement d'un enginyer informàtic en aquest camp.

En aquest treball, les matemàtiques ens ajudaran a definir el comportament i les limitacions del nostre simulador, per a poder configurar-lo amb coneixement a posteriori, i sempre ser conscients de l'estat del sistema que s'està processant.

És a dir, el problema a resoldre, simplificadament, consisteix a computar milers d'interaccions de forces, que segueixen un model físic o matemàtic concret, de manera molt eficient gràcies a certes aproximacions i generalitzacions, amb l'objectiu final d'obtenir una simulació visualment realista del comportament d'un objecte sòlid deformable o fluid.

1.1.1. Conceptes bàsics

Abans d'entrar en detalls, hi ha uns coneixements bàsics que s'han de tenir en compte degut a l'àmplia projecció dels simuladors de fluids.

Tipus de simuladors

Podem diferenciar els tipus de simuladors en 3 subgrups, que varien intrínsecament en el mateix concepte de la representació del fluid i els algorismes emprats. Aquesta representació marcarà la manera d'interactuar i tractar el fluid tant amb si mateix, com amb cossos externs.

Simuladors en Graella Podem representar el fluid o sòlid en un instant de temps concret com a una magnitud en un punt d'una graella, identificant la quantitat de fluid en aquella posició en un moment concret; per altra banda, cada cel·la de la graella té alguna representació de direcció i magnitud de moviment del fluid, per a poder computar el desplaçament d'aquest.

A més, podem emmagatzemar altra informació a cada posició de la graella, o usar les dades ja guardades per a processar l'estat del sistema en un instant de temps posterior, tenint en compte que a cada cel·la només hi pot haver una quantitat màxima de fluid (volum màxim) perquè no sigui un fluid comprímbil, i així provocar una dissipació d'aquest i el consegüent moviment.

En conjunt, creem un espai acotat per les dimensions de la graella on el fluid es mou de manera quantitativa a través de les diferents cel·les, d'acord amb la mateixa informació d'aquestes.

Aquesta representació es coneix com Euleriana.

Simuladors de Partícules Podem representar un petit volum de material com a una partícula en una posició determinada en un instant de temps, de manera que sempre representa la mateixa quantitat de material, i a més guardem tota la informació de moviment (força, velocitat, ...) a un nivell molt concret representat per la mateixa partícula.

És important entendre que les partícules no representen el cos o fluid a escala de molècules o àtoms, sinó que cada partícula representa una porció contínua del material, o un subconjunt del domini físic a simular.

Aquesta representació permet, per exemple, barrejar dos fluids mantenint alhora les seves propietats separades, o simular interaccions entre partícules o sòlids externs amb un major nivell de detall. També cal dir, que aquest mètode dificulta la tasca de mantenir un volum constant del fluid, i sobretot necessita calcular interaccions entre totes les partícules, còmput que resulta molt costós.

Els simuladors de partícules són coneguts com a Lagrangians.

Simuladors Híbrids Enllaçant els dos conceptes anteriors, obtenim un simulador que, podem dir, rep el millor dels dos mètodes.

Per una banda, es representa el fluid com partícules en cada instant de temps amb totes les seves propietats; i per altra banda el còmput d'interaccions entre aquestes el gestionem mitjançant una graella (la qual defineix l'espai del sistema) a on traspassem les característiques de les partícules en determinades zones, de manera que la computació de col·lisions i/o interaccions és molt més eficient i acotada.

Material Point Method

Un dels mètodes que ara ha ressorgit per a la simulació de materials deformables és el conegut *Material Point Method*, proposat per Sulsky [1], basat en partícules Lagrangianes i una graella Euleriana de rerefons. Aquest mètode és usat donada la seva alta eficiència de còmput, i la possibilitat de detallar la seva precisió amb la mida de la graella.

Aquesta tècnica intenta emmagatzemar les dades de les partícules interpolades en les cel·les més properes, du a terme els càlculs en la graella, i finalment es fa la interpolació de manera inversa i es produeix el desplaçament de la partícula.

Tot i això *MPM* té alguns problemes, ja que per culpa de la simplificació que es produeix a la graella no som capaços de representar velocitats discontinües, o col·lisions molt concretes amb objectes de menor resolució de la graella.

1.1.2. Formulació del problema

Podríem considerar que hi ha dos tipus de simulacions: les que tenen com a objectiu emular la realitat, i les que busquen obtenir una animació visualment realista.

Les simulacions que imiten la realitat tendeixen a requerir moltíssima precisió i algorismes poc optimitzables, les quals acostumen a necessitar ser processades en un supercomputador per obtenir resultats adequats en un temps raonable.

Per altra banda, la simulació amb objectius només visuals (com és el nostre cas) intenta ser ràpida, visualment realista i, sobretot, configurable per a poder triar quins resultats es volen aconseguir. Encara que no s'intenti emular la realitat al cent per cent, és important imitar-la al màxim acceptant generalitzacions i truncaments per a accelerar la velocitat de computació; depenent de la situació podríem simplificar el model el suficient per arribar a tenir una simulació en temps real.

A causa del fet que es busca trobar una fidel i ràpida aproximació a la realitat, no existeix una solució analítica (ni general ni específica) per a produir aquestes simulacions. També s'ha de tenir en compte quina és la versatilitat de la simulació, perquè el mateix algorisme implementat limitarà les propietats del fluid o sòlid. Finalment també hi ha la possibilitat, i el problema, de fer interaccionar el fluid a simular amb algun cos extern (tant en moviment com immòbil) i la transmissió de forces entre aquests.

Així doncs, aquest treball pretén estudiar com plantejar solucions per a implementar les qüestions anteriors, i alhora aplicar-les per a conèixer les limitacions i habilitats de cadascuna, en un determinat sistema; i també servir com a base introductòria a aquesta temàtica tan dispersa i tècnica a qualsevol lector.

1.2. Estat de l'art

La simulació de materials deformables és una àmplia àrea d'estudi des de fa diverses dècades, que s'ha anat dividint en múltiples tècniques i cada branca d'estudi va aprofundint en paral·lel de la resta.

En aquest treball estudiarem particularment el *Material Point Method*, o *MPM*, que neix de la generalització de *Particle in Cell* (PIC) i *Fluid Implicit Particle Method* (FLIP) [1]. Hi ha hagut nombrosos i extensos treballs de FLIP en els darrers anys [3, 4], mentre que *MPM* ha sigut realment introduït i començat a estudiar recentment perquè, al contrari que FLIP, aquest pot tractar amb fluids i sòlids comprimibles de manera estable.

Així doncs, *MPM* s'està començant a estudiar i a usar per a simular diferents tipus d'interaccions innovadores. Com s'ha dit abans, destaca la simulació de neu per la pel·lícula de *Frozen* de Disney [5] o la interacció entre o amb múltiples materials deformables [6], per posar alguns exemples. Es pot trobar més informació dels treballs en *MPM* a [7].

Just recentment, l'any 2018, ha sorgit una nova tècnica per a resoldre *MPM*, anomenada *MLS-MPM* (*Moving Least Squares Material Point Method*) [8] que, generalment, busca millors aproximacions a la graella gràcies a interpolacions amb *B-splines*, i l'ús de *MLS* per a resoldre les corresponents equacions diferencials de manera simplificada.

1.3. Estructura del document

Aquest document està dividit en diverses parts ben diferenciades, les quals intenten ser independents en la mesura del possible, de manera que es puguin seguir els continguts d'aquest document segons la motivació del lector a llegir els diferents apartats.

Primerament, en aquest mateix capítol 1, s'introdueix el problema a resoldre en aquest treball, així com la motivació i l'abast d'aquest.

S'encoratja la lectura de la següent secció 1.4 per entendre tota la notació matemàtica emprada en aquesta memòria, per evitar confusions en els següents capítols.

Seguidament trobem els capítols 2 i 3, centrats en la teoria darrere les diverses parts a tenir en compte durant la simulació de cossos deformables. Aquests són matemàticament intensos, tot i que intenten ser adequadament progressius fins a arribar a definir les bases del sistema a resoldre.

En el capítol 4 es descriu la tècnica del *material point method* (*MPM*), la qual és central en aquest treball, a partir dels continguts dels anteriors capítols i introduint altres tècniques aplicables.

Seguidament, en el capítol 5, s'implementa un software capaç de dur a terme simulacions de materials deformables seguint la tècnica de *MPM*, de manera molt personalitzable. Hi ha codi disponible i consultable en tot moment.

Per altra banda, en el capítol 6 es descriu tot el *software* perifèric al simulador, així com un visualitzador gràfic, un gestor de carrega i guardat de simulacions, i l'exportació d'aquestes.

Per resumir els resultats del treball, el capítol 7 mostra diverses simulacions obtingudes, amb l'objectiu d'exposar la major quantitat de diferents comportaments que el *software* és capaç de simular.

En el penúltim capítol 8 del treball, se'n descriu l'organització d'aquest, juntament amb una teòrica gestió econòmica i un estudi de sostenibilitat.

Finalment, en l'últim capítol 9, es donen les conclusions d'aquest treball de fi de grau, així com s'expressa el possible treball futur en aquest.

Cal afegir, que hi ha 3 apèndixs adjunts en aquest document.

Un primer apèndix A, que extreu totes aquelles derivacions matemàtiques que no són essencials d'aquest projecte, però serveixen per aprofundir en la simulació de materials deformables.

En segon lloc es troba l'apèndix B, el qual llista totes les simulacions disponibles computades amb el programa dut a terme, així com enllaços als recursos i descripcions d'aquestes.

I en l'últim apèndix C, s'hi troben extraccions de codi que poden resultar interessants durant la lectura d'aquesta memòria.

1.4. Sobre la notació

En aquest document, i sobretot en els capítols més matemàticament intensos, s'ha establert una notació i tipografia per poder distingir els diferents elements de les equacions i fórmules.

La taula 1.1 resumeix les notacions matemàtiques que usaré en els següents apartats. Notar que hi ha dues excepcions per les coordenades *Eulerianes* i *Lagrangianes*, \vec{x} i \vec{X} respectivament, les quals es marquen en negreta per denotar la seva importància, i s'explicaran en profunditat en la secció 2.1.

Tipus	Notació	Exemples
Escalar	itàlica	a, t, v_x, Ψ
Vector o punt	vector	\vec{v}, \vec{u}
Vector normalitzat	barret	\hat{n}, \hat{e}_1
Matriu	negreta	$\mathbf{F}, \boldsymbol{\sigma}$

Taula 1.1: *Nomenclatura*. Resum de la nomenclatura usada en la memòria. (Elaboració pròpia)

Tots els vectors, tant en 2 com en 3 dimensions, són vectors columna.

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \quad (1.1)$$

Els quals podem transformar en vectors fila mitjançant la transposició

$$\vec{v}^T = (v_x \quad v_y \quad v_z) \quad (1.2)$$

Les operacions sobre vectors i matrius es defineixen en la secció [A.1](#) de càlcul vectorial.

En el cas de les matrius, l'accés dels elements escalars m_{ij} de la matriu \mathbf{M} , estan indexats de manera que $1 \leq i, j \leq 3$, on i denota la columna i j la fila.

$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \quad (1.3)$$

Finalment, a la taula [1.2](#) es pot trobar un resum de les abreviatures més recurrents en aquest document.

Abreviatura	Significat
\vec{x}	Descriptor <i>Eulerià</i> o espacial
\vec{X}	Descriptor <i>Lagrangià</i> o material
\vec{x}_i	Posició de la cel·la i
\vec{x}_p	Posició de la partícula p
$\phi(\vec{X}, t)$	Funció de mapeig de descripció material a espacial
\mathbf{F}	Gradient de deformació
$\mathbf{F}_E, \mathbf{F}_P$	Descomposició elàstica i plàstica del gradient de deformació
J	Determinant de \mathbf{F}
ρ	Densitat
$\boldsymbol{\sigma}$	Tensor de tensió de <i>Cauchy</i>
\mathbf{P}	Primer tensor de tensió de <i>Piola-Kirchhoff</i>
\mathbf{S}	Segon tensor de tensió de <i>Piola-Kirchhoff</i>
$\boldsymbol{\epsilon}$	Tensor d'esforç
E	Modul de Young
ν	Ràtio de Poisson
λ, μ	Coefficients de Lamé
\mathcal{E}	Enduriment
$\vec{t}^{(\hat{n})}$	Tracció en una direcció \hat{n}
$\Psi(\mathbf{F})$	Funció escalar, energètica de densitat d'esforç.
$N_i(\vec{x})$	Funció d'interpolació centrada a \vec{x}_i , basada en una B-spline
w_{ip}	Escalar corresponent a $N_i(\vec{x}_p)$
$\mathbf{C}_p^n, \mathbf{D}_p^n$	Tensors d' <i>APIC</i> , de la partícula p en l'instant n

Taula 1.2: *Llistat d'abreviatures.* Abreviatures i notació de tots els termes en aquesta memòria. (Elaboració pròpia)

2. Deformacions

En aquest capítol es presenta el rerefons necessari per a entendre com s'emmagatzemen, tracten, i utilitzen les diverses deformacions que aquest treball procura.

De ser necessària una introducció o un recordatori de càlcul vectorial, així com a les seves operacions més comunes, consultar [A.1](#).

2.1. Cinemàtica

A l'hora de simular qualsevol tipus de deformació, d'alguna manera o una altra, acabem emulant un moviment. Necessitem un cert formalisme per descriure tots els tipus de moviment que patirà el nostre cos o fluid. També cal afegir que dins de la cinemàtica entra la deformació, i per tant també descriurem com funciona el deformar una secció continua del material.

D'aquesta manera, durant el següent apartat tractarem cada porció del material com una secció continua de matèria, tant si es tracta d'un sòlid, líquid o gas, definint quantitats com densitat, velocitat, forces... per cadascuna d'aquestes porcions. Aquest fet ens evita haver de tractar en l'estudi a nivell microscòpic.

El següent anàlisi segueix molt de prop al de Bonet [14], tot i que una abstracció més simplificada es troba al curs d'MPM de *SIGGRAPH* [7].

2.1.1. Descripció del moviment continu

Per marcar que un cos ha experimentat una deformació d'algun tipus necessitem una referència per a comparar la deformació. Com que estem tractant amb un material continu, seria molt difícil definir una deformació en un instant t d'acord amb el mateix cos en l'instant anterior $t - \epsilon$, amb $t - \epsilon > 0$, per qualsevol ϵ . Per això tota deformació es marcarà d'acord a un estat inicial del cos en temps $t = 0$.

Aquesta deformació elàstica, o moviment, és du a terme a través d'una funció ϕ , com mostra la figura 2.1. Aquesta funció mapeja un seguit de porcions del material, identificades per coordenades cartesianes $\vec{\mathbf{X}}$, sobre la base \mathbf{E} en el temps inicial $t = 0$, cap una coordenada alternativa $\vec{\mathbf{x}}$, en una base \mathbf{e} , en un altre instant de temps. Per tant podem definir aquesta funció del moviment com a:

$$\vec{\mathbf{x}} = \phi(\vec{\mathbf{X}}, t) \quad (2.1)$$

És a dir, la porció de material identificada amb la coordenada $\vec{\mathbf{X}}$, després d'un temps t queda desplaçada a la coordenada $\vec{\mathbf{x}}$. Generalment les bases \mathbf{E} i \mathbf{e} seran equivalents, però s'ha de tenir en compte que és possible dur a terme aquest canvi d'espai amb la funció ϕ .

També es pot entendre la funció ϕ com la transformació del cos sense deformar i el deformat, i per tant $\vec{\mathbf{X}} = \phi(\vec{\mathbf{X}}, 0)$.

És important veure que també existeix la inversa de la funció material, expressada com a ϕ^{-1} , ja que ϕ és (en teoria) bijectiva.

$$\forall \vec{\mathbf{X}}, \forall t, \exists \vec{\mathbf{x}} : \vec{\mathbf{X}} = \phi^{-1}(\vec{\mathbf{x}}, t) \quad (2.2)$$

Per tant, en tot moment una partícula pertany a una regió de l'espai.

2.1.2. Descripció del material

Per descriure un material deformable hem d'emmagatzemar diferents quantitats, com la massa o la velocitat, d'acord a les diferents porcions d'aquest. Això ho podem fer de dues maneres: o respecte l'estat del cos abans de la deformació, o respecte l'estat durant la deformació.

Si descrivim el cos respecte l'estat del cos abans de la deformació, estem usant una descripció material o *Lagrangiana*, de manera que fem un seguiment de, posant un exemple a escala microscòpica, les partícules del material; o en un cas més abstracte, d'una porció del material.

Per altra banda, podem usar una descripció espacial o *Euleriana* si seguim el comportament a cada posició de l'espai en conjunt.

Podem exemplificar les diferències amb aquestes dues descripcions amb un simple exemple, en que volem extreure la densitat ρ després d'un temps t , donada una funció D que ens retorna aquest valor.

- *Lagrangiana*: La variació de densitat es descriu en funció de la densitat a la coordenada original $\vec{\mathbf{X}}$, respecte l'instant $t = 0$:

$$\rho = D(\vec{\mathbf{X}}, t)$$

- *Euleriana*: La quantitat de densitat que conté una regió de l'espai $\vec{\mathbf{x}}$, que depèn del temps.

$$\rho = D(\vec{\mathbf{x}}, t)$$

Canvis en els valors de densitat indiquen que la regió espacial $\vec{\mathbf{x}}$ és ocupada per una o unes partícules diferents.

Moltes vegades és important obtenir dades en descripció espacial en referència a la coordenada en descripció material, o viceversa, ja que facilita l'obtenció d'aquestes dades. Per exemple, donada una descripció material $\vec{\mathbf{X}}$, quina és la densitat en la regió deformada? Això ho podem fer usant l'equació 2.1, i ens

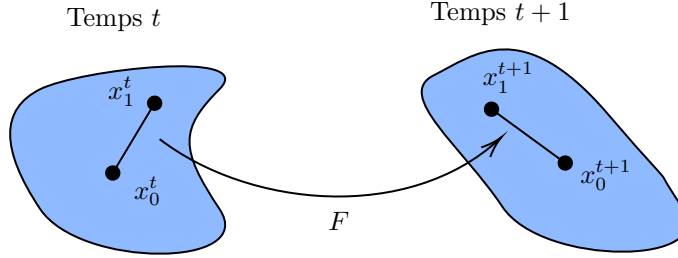


Figura 2.1: *Gradient de deformació*. Mateix cos en dos instants de temps diferents t i $t + 1$ després d'una deformació. Cal veure que, per exemple $\vec{x}_1^t = \phi(\vec{x}_1^0, t)$. (Elaboració pròpia)

dona l'avantatge que és molt més senzill calcular la densitat en un punt espacial.

$$\rho(\vec{X}, t) = \rho(\phi(\vec{X}, t), t) \quad (2.3)$$

2.1.3. Gradient de deformació

El gràdient de deformació \mathbf{F} , és un dels elements més importants a l'hora de definir una deformació; o més bé: a com produir la deformació.

Aquest és un tensor que representa, localment, la deformació d'un material.

Suposem que tenim dos punts \vec{x}_0^t i \vec{x}_1^t molt propers en un instant de temps concret t , i que en un moment de temps posterior el material s'ha deformat i els mateixos punts ara es troben a \vec{x}_0^{t+1} i a \vec{x}_1^{t+1} respectivament, tal i com mostra la figura 2.1.

La idea del gradient de deformació és que és un tensor que descriu el canvi de configuració respecte els propis punts; de manera que $(\vec{x}_1^{t+1} - \vec{x}_0^{t+1}) = \mathbf{F}(\vec{x}_1^t - \vec{x}_0^t)$.

El gradient de deformació es defineix:

$$\mathbf{F}(\vec{X}, t) = \frac{\partial \phi(\vec{X}, t)}{\partial \vec{X}} \quad (2.4)$$

Una altra particularitat del gradient de deformació és que descriu els canvis de volum infinitesimals. Aquesta dada es pot extreure del mateix determinant de \mathbf{F} , determinant usualment anomenat la Jacobiana i denotat amb J , $J = \det(\mathbf{F})$, expressant la variació infinitesimal de volum entre els instants de temps t i $t + 1$. Per rotacions i translacions $J = 1$, $J > 1$ indica un increment de volum i $J < 1$ un decrement. En el rar cas de $J = 0$ el volum s'ha tornat zero, potser degut a una compressió. Finalment en el cas $J < 0$ el material s'inverteix.

De manera que donat un volum original V_0 i un volum després de la deformació V :

$$dV = J \cdot dV_0 \quad (2.5)$$

També podem expressar el següent:

$$d\vec{x} = \mathbf{F} \cdot d\vec{X} \quad (2.6)$$

2.2. Derivada del gradient de deformació

Per poder computar la derivada del gradient de deformació en funció del temps hem de fer el següent:

$$\frac{\partial}{\partial t} \mathbf{F}(\vec{X}, t) = \frac{\partial}{\partial t} \frac{\partial \phi}{\partial \vec{X}}(\vec{X}, t) \quad (2.7)$$

$$= \frac{\partial \vec{v}}{\partial \vec{X}}(\vec{X}, t) \quad (2.8)$$

$$= \frac{\partial \vec{v}}{\partial \vec{X}}(\phi(\vec{X}, t), t) \quad (2.9)$$

$$= \frac{\partial \vec{v}}{\partial \phi(\vec{X}, t)} \frac{\partial \phi}{\partial \vec{X}}(\vec{X}, t) \quad (2.10)$$

$$= \frac{\partial \vec{v}}{\partial \vec{x}} \cdot \mathbf{F} \quad (2.11)$$

On primer hem descompost el gradient amb l'equació 2.4 i transformat en velocitat *Lagrangiana* gràcies a la derivada temporal (una derivació de la velocitat més en profunditat és consultable a A.36). Seguidament fem el *pull back* de la velocitat *Euleriana* i duem a terme la derivada multivariable (definició consultable a l'equació A.44). Finalment es transformen els termes *Eulerians* i el gradient.

Veiem com el gradient de deformació depèn del gradient de la velocitat *Euleriana*, molt semblant a la derivada material, concepte important en simulació, tractat a l'apèndix A.5.2.

Podem reescriure-ho com a:

$$\frac{\partial}{\partial t} \mathbf{F}(\vec{X}, t) = \nabla \vec{v} \cdot \mathbf{F} \quad (2.12)$$

2.3. Tensors de deformació

Per a definir una deformació usem usualment el gradient de deformació \mathbf{F} definit a la secció 2.1.3; però aquest tensor ens dóna informació de més que voldrem disseccionar.

El que voldríem extreure del gradient \mathbf{F} són les rotacions pures, ja que aquestes no introdueixen cap tipus d'esforç al material (Consultable a [A.6](#)).

Això és molt senzill de realitzar, ja que una rotació \mathbf{R} seguida de la seva inversa \mathbf{R}^{-1} equival a cap rotació, i la inversa d'una matriu de rotació és equivalent a la seva transposada, és a dir: $\mathbf{R}\mathbf{R}^{-1} = \mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{R}^{-1}\mathbf{R} = \mathbf{I}$. Per tant podem extreure la rotació de \mathbf{F} multiplicant per la seva transposada.

Aquest tensor s'anomena el tensor dret de deformació de *Cauchy-Green* \mathbf{C} :

$$\mathbf{C} = \mathbf{F}^T \mathbf{F} \quad (2.13)$$

I l'aplicabilitat d'aquest tensor la veurem al següent apartat.

Per altra banda també podem definir el tensor esquerre de *Cauchy-Green* \mathbf{B} com:

$$\mathbf{B} = \mathbf{F} \mathbf{F}^T \quad (2.14)$$

Finalment podem comprovar que s'elimina realment la rotació, si descomponem el gradient \mathbf{F} de manera polar per la dreta (descomposició polar consultable a [A.3](#)):

$$\mathbf{F}^T \mathbf{F} = (\mathbf{R} \cdot \mathbf{U})^T \cdot \mathbf{R} \cdot \mathbf{U} = \mathbf{U}^T \cdot \mathbf{R}^T \cdot \mathbf{R} \cdot \mathbf{U} = \mathbf{U}^T \cdot \mathbf{U} \quad (2.15)$$

És important remarcar que d'aquest tensor els valors de la diagonal λ_1^2 , λ_2^2 i λ_3^2 , marquen les ràtios de deformació en cadascun dels vectors propis del tensor \mathbf{F} . I per a materials incompressibles $\lambda_1 \lambda_2 \lambda_3 = 1$.

2.4. Tensió

Per a caracteritzar o quantificar una deformació, podem usar l'esforç ϵ . Una mesura adimensional que identifica l'elongació i compressió relativa en un material. Aquest esforç es pot considerar senzillament lineal, o se'n poden utilitzar els seus vessants de dimensions superiors, que atorguen més precisió. Les derivacions de l'esforç no lineal es troben a l'apèndix [A.6](#).

A partir d'aquest esforç, seria útil computar una quantitat física que expressés quina és la distribució de forces internes del material, producte de la seva deformació, per a usar en la simulació. Aquesta força interna s'anomena tensió, i tracció a la seva distribució.

2.4.1. Tensió de *Cauchy*

Generalment la tensió de *Cauchy* es defineix com a $\sigma = F/A$ (N/m^2 en unitats del SI), i com que es tracta d'un tensor es pot visualitzar com la pressió efectuada sobre cadascuna de les seccions del material. Aquest inclou no només forces uniaxials, sinó també forces de tensió tangencial (o de fregament).

Més concretament, el tensor identifica en la seva diagonal les tensions normals σ , i en el triangle superior i inferior les tensions tangencials τ , de manera simètrica.

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_z \end{bmatrix} \quad (2.16)$$

Podem establir una relació directa entre la tensió i l'esforç mitjançant el mòdul de *Young* E , que defineix la rigidesa d'un material davant deformacions. Amb aquesta constant podem calcular, de manera uniaxial i lineal, quina és la tensió resultant d'un esforç.

$$\sigma = E \cdot \epsilon \quad (2.17)$$

De manera que E té les mateixes unitats que la tensió, i varia de material a material.

Per materials anisotròpics, la constant E es transforma en una funció direccional.

2.4.2. Tracció superficial

Per altra banda recordem que el tensor de *Cauchy* representa una operació definida sobre el material deformat (Coordenades espacials \vec{x}). Definim \vec{t} el vector de tracció resultant d'aplicar la tensió en una direcció \hat{n} , més concretament les forces \vec{f} actuant sobre una superfície deformatada s , amb normal \hat{n} (vector unitari). La primera definició orientativa que podem donar és:

$$\vec{t}^{(\hat{n})} = \frac{d\vec{f}}{ds} \quad (2.18)$$

És a dir, la tracció superficial és un vector que indica la densitat de forces a la superfície d'un material o cos.

Per a trobar una manera de computar aquest vector, emularem una secció infinitesimal d'un material o cos que ha sigut deformat, de manera que ens trobem en un espai descrit *Eulerianament*.

Aquesta secció és el tetraedre de la figura 2.2, amb les tres cares orientades en els plans de coordenades de la base de l'espai, i amb la base orientada en la direcció \hat{n} .

El primer pas consisteix a parametritzar la nostra superfície, definida per la normal \hat{n} , segons els vectors de la base de l'espai \hat{e}_1 , \hat{e}_2 i \hat{e}_3 , que assumim normalitzats sempre.

$$\hat{n} = \begin{pmatrix} \cos(\hat{n}, \hat{e}_1) \\ \cos(\hat{n}, \hat{e}_2) \\ \cos(\hat{n}, \hat{e}_3) \end{pmatrix} = \begin{pmatrix} \hat{n} \cdot \hat{e}_1 \\ \hat{n} \cdot \hat{e}_2 \\ \hat{n} \cdot \hat{e}_3 \end{pmatrix} = \begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix} \quad (2.19)$$

On $\cos(\vec{a}, \vec{b})$ indica el cosinus de l'angle més petit entre els vectors \vec{a} i \vec{b} .

Per altra banda, també ens interessa parametritzar la superfície ds en funció de la base. Recordem que podem projectar qualsevol àrea a un pla mitjançant el cosinus entre les normals de les dues superfícies.

$$\begin{aligned} ds_1 &= n_1 \cdot ds \\ ds_2 &= n_2 \cdot ds \\ ds_3 &= n_3 \cdot ds \end{aligned} \quad (2.20)$$

Finalment podem establir, gràcies a la segona llei de Newton $\sum \vec{f} = m \cdot \vec{a}$, que la suma de totes les forces que actuen sobre el cos serà proporcional a la tracció

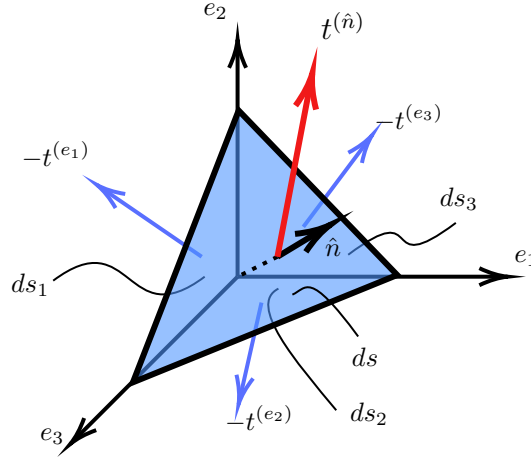


Figura 2.2: Vectors de tracció superficial sobre un tetraedre. Per simplicitat s'ha omès la notació vectorial. Base orientada en direcció \hat{n} , i les tres cares (no visibles) recolzades en els plans de coordenades. Els vectors e_1 , e_2 i e_3 indiquen la base de l'espai. Les fletxes blaves indiquen la tracció en cadascuna de les cares ocultes $-t^{(e_i)}$, i en vermell la tracció resultant en la direcció \hat{n} . Els escalars ds_i indiquen la superfície de cadascuna de les cares. (Elaboració pròpia)

per la superfície de la respectiva cara:

$$\sum \vec{f} = \vec{t}^{(\hat{n})} ds - \vec{t}^{(\hat{e}_1)} ds_1 - \vec{t}^{(\hat{e}_2)} ds_2 - \vec{t}^{(\hat{e}_3)} ds_3 = m \cdot \vec{a} \quad (2.21)$$

$$= \rho \cdot V \cdot \vec{a} \quad (2.22)$$

$$= \rho \cdot \frac{h \cdot ds}{3} \cdot \vec{a} \quad (2.23)$$

La part dreta de l'equació, transforma la massa m en el producte de densitat ρ per volum V del tetraedre. El volum del tetraedre pot ser computat com $V = \frac{1}{3} \cdot A_0 \cdot h$, on A_0 és l'àrea de la base, i h la distància des de la base fins al vèrtex oposat.

Amb aquesta informació podem transformar totes les superfícies ds_i en funció de ds usant l'equació 2.20:

$$\vec{t}^{(\hat{n})} ds - \vec{t}^{(\hat{e}_1)} (n_1 \cdot ds) - \vec{t}^{(\hat{e}_2)} (n_2 \cdot ds) - \vec{t}^{(\hat{e}_3)} (n_3 \cdot ds) = \rho \cdot \frac{h \cdot ds}{3} \cdot \vec{a} \quad (2.24)$$

I per tant podem eliminar completament el terme ds :

$$\vec{t}^{(\hat{n})} - \vec{t}^{(\hat{e}_1)} n_1 - \vec{t}^{(\hat{e}_2)} n_2 - \vec{t}^{(\hat{e}_3)} n_3 = \rho \cdot \frac{h}{3} \cdot \vec{a} \quad (2.25)$$

Una important observació és que el tetraedre és infinitesimalment petit, de manera que les seves mides són nul·les; això ens serveix per suposar en el seu límit $h \rightarrow 0$, de manera que la part dreta de l'equació s'aproxima a 0. Podem llavors reescriure l'equació anterior:

$$\vec{t}^{(\hat{n})} = \vec{t}^{(\hat{e}_1)} n_1 + \vec{t}^{(\hat{e}_2)} n_2 + \vec{t}^{(\hat{e}_3)} n_3 \quad (2.26)$$

Recordem que estem usant vectors columna, però per convenció la tensió s'acostuma a expressar en vectors fila, de manera que és més correcte expressar aquesta relació:

$$\vec{t}^{(\hat{n})T} = \vec{t}^{(\hat{e}_1)T} n_1 + \vec{t}^{(\hat{e}_2)T} n_2 + \vec{t}^{(\hat{e}_3)T} n_3 \quad (2.27)$$

Que la podem reescriure com a una operació matricial:

$$\vec{t}^{(\hat{n})T} = \begin{pmatrix} n_1 & n_2 & n_3 \end{pmatrix} \cdot \begin{bmatrix} \vec{t}^{(\hat{e}_1)T} \\ \vec{t}^{(\hat{e}_2)T} \\ \vec{t}^{(\hat{e}_3)T} \end{bmatrix} \quad (2.28)$$

Si ens fixem, la matriu de la dreta és un tensor que opera sobre una direcció; justament aquesta matriu és el tensor transposat de tensió de *Cauchy*:

$$\vec{t}^{(\hat{n})T} = \hat{n}^T \cdot \boldsymbol{\sigma}^T \quad (2.29)$$

Amb aquest tensor podem computar, donat un material, la tracció en qualsevol direcció del cos deformat; equació que resulta molt útil per controlar la distribució de forces d'un sòlid elàstic.

Per ser consistents, podem expressar el mateix en vectors columna transposant ambdós costats de l'equació (recordar que $\boldsymbol{\sigma}$ és simètrica):

$$\vec{t}^{(\hat{n})} = \boldsymbol{\sigma} \cdot \hat{n} = \boldsymbol{\sigma}^T \cdot \hat{n} \quad (2.30)$$

2.4.3. Tensors de tensió de *Piola-Kirchhoff*

La tensió de *Cauchy* $\boldsymbol{\sigma}$, que hem vist fins ara, representa les forces internes del material respecte la configuració deformatada. Per altra banda, seria interessant poder representar aquestes forces respecte d'una configuració de referència, tal com ho fa el gradient de deformació \mathbf{F} .

Concretament existeixen dos tensors de *Piola-Kirchhoff*, que descriuen cadascuna de les configuracions, respectivament:

Primer tensor de *Piola-Kirchhoff*

El primer tensor de *Piola-Kirchhoff* \mathbf{P} , defineix el tensor de tensió relacionant la descripció espacial amb la descripció del model; és a dir, relaciona forces de la descripció *Euleriana* amb superfícies de la *Lagrangiana*, amb un tensor $\vec{T}^{(\hat{N})}$.

$$\vec{T}^{(\hat{N})} = \frac{d\vec{f}}{dS} \quad (2.31)$$

Aquest tensor ens permet obtenir la tracció superficial $\vec{T}^{(\hat{N})}$, en una direcció \hat{N} , en descripció *Lagrangiana*. De manera anàloga al tensor de *Cauchy*, podem obtenir el símil de l'equació 2.30 però per aquest primer tensor de *Piola-Kirchhoff*:

$$\vec{T}^{(\hat{N})} = \mathbf{P} \cdot \hat{N} \quad (2.32)$$

Ara bé, per calcular el tensor \mathbf{P} a partir d'informació de la deformació, cal recordar algunes transformacions:

$$d\vec{x} = \mathbf{F}d\vec{X} \quad (2.6 \text{ revisitada})$$

$$dV = J \cdot dV_0 \quad (2.5 \text{ revisitada})$$

On $J = \det(\mathbf{F})$, V_0 el volum abans de la deformació i V el volum deformat. D'aquesta manera podem definir les transformacions més elementals usant informació que ja coneixem. Ara bé, cal afegir algunes equacions més per transformar el volum en funció de la superfície del cos, en descripció material S i espacial s .

Primer de tot introduïm el terme superfície orientada, que indica la superfície com a magnitud del vector direcció:

$$d\vec{s} = ds \cdot \hat{n} \quad (2.33)$$

$$d\vec{S} = dS \cdot \hat{N} \quad (2.34)$$

Veure que \hat{n} i \hat{N} indiquen la normal del pla infinitesimal de la mateixa superfície presa, respectivament de la seva descripció.

Així podem definir els volums en funció de petites variacions en cadascuna de les coordenades $d\vec{x}$ i $d\vec{X}$:

$$dV = d\vec{s}^T \cdot d\vec{x} \quad (2.35)$$

$$dV_0 = d\vec{S}^T \cdot d\vec{X} \quad (2.36)$$

Amb tota aquesta descomposició, i recordant que volem obtenir alguna manera de representar la tensió respecte del cos de referència. A partir de l'equació 2.36, substituïm dV per l'equació 2.5, i dV_0 per l'equació 2.35; finalment dividim per $d\vec{X}$.

$$J \cdot dV_0 = d\vec{s}^T \cdot d\vec{x} \quad (2.37)$$

$$J \cdot d\vec{S}^T \cdot d\vec{X} = d\vec{s}^T \cdot d\vec{x} \quad (2.38)$$

$$J \cdot d\vec{S}^T \cdot d\vec{X} = d\vec{s}^T \cdot \mathbf{F} \cdot d\vec{X} \quad (2.39)$$

$$J \cdot d\vec{S}^T = d\vec{s}^T \cdot \mathbf{F} \quad (2.40)$$

Reordenant l'equació podem obtenir la diferencial de l'àrea deformada, en funció de variables del cos de referència:

$$d\vec{s}^T = J \cdot d\vec{S}^T \cdot \mathbf{F}^{-1} \quad (2.41)$$

Que pot ser escrit també, transposant ambdós costats de l'equació, com:

$$d\vec{s} = J \cdot \mathbf{F}^{-T} \cdot d\vec{S} \quad (2.42)$$

Aquesta equació és coneguda com la relació de *Nanson* [15].

Ara bé, per aplicar aquesta última equació hem de retornar a l'objectiu original, que és obtenir el tensor \mathbf{P} , i ara disposem d'una relació entre superfícies direccionals, i recordem que sabem computar la tracció:

$$\vec{t}^{(\hat{n})} = \boldsymbol{\sigma} \cdot \hat{n} \quad (2.30 \text{ revisitada})$$

$$\vec{T}^{(\hat{N})} = \mathbf{P} \cdot \hat{N} \quad (2.32 \text{ revisitada})$$

Recordem també que la tracció relaciona superfície amb força $d\vec{f} = ds \cdot \vec{t}^{(\hat{n})}$ (Equació 2.18), una força que és independent de l'estat de referència usat, per tant podem multiplicar per les respectives superfícies les anteriors equacions, de manera que equivalguin a la mateixa força:

$$d\vec{f} = \vec{t}^{(\hat{n})} \cdot ds = \boldsymbol{\sigma} \cdot \hat{n} \cdot ds \quad (2.43)$$

$$d\vec{f} = \vec{T}^{(\hat{N})} \cdot dS = \mathbf{P} \cdot \hat{N} \cdot dS \quad (2.44)$$

Que podem reescriure usant les superfícies direccionals $d\vec{s}$ i $d\vec{S}$:

$$d\vec{f} = \vec{t}^{(\hat{n})} \cdot ds = \boldsymbol{\sigma} \cdot d\vec{s} \quad (2.45)$$

$$d\vec{f} = \vec{T}^{(\hat{N})} \cdot dS = \mathbf{P} \cdot d\vec{S} \quad (2.46)$$

D'aquesta manera podem relacionar les dues equacions de la tracció superficial de *Cauchy* i de *Piola-Kirchhoff*:

$$\boldsymbol{\sigma} \cdot d\vec{s} = \mathbf{P} \cdot d\vec{S} \quad (2.47)$$

A partir d'aquí podem eliminar el terme $d\vec{s}$ si el substituïm per la relació de *Nanson* (Equació 2.42).

$$\boldsymbol{\sigma} \cdot J \cdot \mathbf{F}^{-T} \cdot d\vec{S} = \mathbf{P} \cdot d\vec{S} \quad (2.48)$$

Finalment eliminem els termes comuns dividint per $d\vec{S}$, i obtenim finalment la fórmula del primer tensor de tensió de *Piola-Kirchhoff*:

$$\mathbf{P} = J \cdot \boldsymbol{\sigma} \cdot \mathbf{F}^{-T} \quad (2.49)$$

Aquesta també ens permet recalculer el tensor de *Cauchy*, de ser necessari:

$$\boldsymbol{\sigma} = \frac{1}{J} \cdot \mathbf{P} \cdot \mathbf{F}^T \quad (2.50)$$

És important veure que, a diferència del tensor de tensió de *Cauchy*, el tensor \mathbf{P} no és simètric, ja que $\mathbf{P} = J \cdot \boldsymbol{\sigma} \cdot \mathbf{F}^{-T} \neq J \cdot \mathbf{F}^{-1} \cdot \boldsymbol{\sigma}^T = \mathbf{P}^T$.

Segon tensor de *Piola-Kirchhoff*

Per altra banda, el segon tensor de *Piola-Kirchhoff* \mathbf{S} és parametriztat només per coordenades materials (*Lagrangianes*) de manera que caracteritza les forces en el cos de referència en funció de superfícies del mateix cos de referència, dS .

Aquest tensor és important en la simulació, però no s'aplica en aquest treball. La seva derivació és consultable a l'apèndix A.7, així com les relacions tant amb el primer tensor de *Piola-Kirchhoff* i el de *Cauchy*.

2.5. Hiperelasticitat

Un material de *Green* o hiperelàstic, és un material que exhibeix propietats elàstiques.

Particularment han de poder ser descrits amb un model constitutiu, o equació constitutiva, relacionant diverses variables del material entre si, aproximant la resposta del material a un cert estímul (extern o intern).

En el cas de l'elasticitat ens interessa modelar, per exemple, la relació entre la tensió i l'esforç de manera no lineal, a diferència de l'equació 2.17, i poder descriure també materials tant isotròpics com anisotròpics, compressibles i incompressibles.

Generalment es compleix que, donat un sòlid hiperelàstic, en podem derivar la seva funció energètica segons el seu gradient de deformació per obtenir el primer tensor de *Piola-Kirchhoff*:

$$\mathbf{P} = \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}) \quad (2.51)$$

2.5.1. Funcions energètiques de densitat d'esforç

També anomenades en anglès *strain energy density functions*, són funcions que retornen un valor escalar en funció d'un gradient de deformació \mathbf{F} (Secció 2.1.3), o algun terme semblant com el tensor dret de deformació de *Cauchy-Green* (Secció 2.3)

Aquestes es defineixen amb la lletra W o la lletra grega Ψ , i particularment estableixen una relació amb la quantitat d'energia empleada per a deformar una unitat de volum d'un sòlid, d'acord amb un esforç donat.

D'aquesta manera, generalment, definirem aquestes funcions com a $\Psi(\mathbf{F})$.

Les funcions Ψ són molt diverses i varien segons la propietat en concret que es vol modelar. De totes maneres n'existeixen diverses ja derivades que són aplicables a diversos materials.

2.5.2. *Neo-Hookean*

En aquesta secció definim la funció energètica d'un material compressible *Neo-Hookean* per a deformacions no lineals.

Aquesta funció ve definida, com és normal, per un gradient de deformació \mathbf{F} , i per unes constants λ i μ que definiran el comportament d'aquest cos:

$$\Psi(\mathbf{F}) = \frac{\mu}{2}(tr(\mathbf{C}) - d) - \mu \lg(J) + \frac{\lambda}{2} \lg^2(J) \quad (2.52)$$

On recordem que $\mathbf{C} = \mathbf{F}^T \cdot \mathbf{F}$ el tensor dret de deformació de *Cauchy-Green* (Veure 2.3), i $J = \det(\mathbf{F})$. d representa la dimensionalitat del problema (2 o 3). Notar com en el cas que \mathbf{F} no introdueixi cap deformació ($\mathbf{F} = \mathbf{I}$) l'energia elàstica de Ψ és zero.

Aquesta funció energètica es basa en eliminar el component rotacional del gradient de deformació \mathbf{F} i només tractar els estiraments, ja que el tensor \mathbf{C} extreu les rotacions del gradient \mathbf{F} .

Derivant aquesta funció respecte del gradient de deformació, obtenim el primer tensor de *Piola-Kirchhoff*, tal com l'equació 2.51 indica.

$$\mathbf{P} = \mu(\mathbf{F} - \mathbf{F}^{-T}) + \lambda \lg(J) \mathbf{F}^{-T} \quad (2.53)$$

Alhora, també podem obtenir el segon tensor de *Piola-Kirchhoff* mitjançant la relació de l'equació A.82:

$$\mathbf{S} = \mu(\mathbf{I} - \mathbf{C}^{-1}) + \lambda \ln(J) \mathbf{C}^{-1} \quad (2.54)$$

I finalment podem obtenir el tensor de tensió de *Cauchy* a través del primer tensor de *Piola-Kirchhoff*, segons marca l'equació 2.50, on $\mathbf{B} = \mathbf{F} \mathbf{F}^T$ és el tensor esquerre de *Cauchy-Green* (Equació 2.14):

$$\boldsymbol{\sigma} = \frac{\mu}{J}(\mathbf{B} - \mathbf{I}) + \frac{\lambda}{J} \ln(J) \mathbf{I} \quad (2.55)$$

Les constants μ i λ no són arbitràries, i s'anomenen els coeficients de *Lamé*. Van lligades al mòdul de *Young* E (Vist a la secció 2.4.1) i al coeficient de *Poasson* ν .

Com s'ha dit anteriorment, el mòdul de *Young* E indica la rigidesa del material en termes d'elasticitat lineal.

Per altra banda, el coeficient de *Poasson* relaciona la deformació en un eix respecte de la deformació en el pla perpendicular; és a dir, l'estrenyiment. Per exemple, en estirar un cilindre elàstic en l'eix x , la longitud del cilindre en aquest eix augmenta, però en el pla perpendicular yz el diàmetre d'aquest disminueix.

$$\nu = -\frac{d\epsilon_{trans}}{d\epsilon_{axial}} \quad (2.56)$$

Veiem que es defineix en funció de l'esforç aplicat en els eixos, segons l'esforç transversal ϵ_{trans} a l'esforç en l'eix de referència ϵ_{axial} .

Així doncs, definim μ i λ en funció d'aquests dos paràmetres del material:

$$\lambda = \frac{E \cdot \nu}{(1 + \nu) \cdot (1 - 2 \cdot \nu)} \quad (2.57)$$

$$\mu = \frac{E}{2 \cdot (1 + \nu)} \quad (2.58)$$

2.5.3. Corrotacional Fix

El model constitutiu corrotacional fix es basa en el fet que un gradient de deformació \mathbf{F} pot ser sempre descompost en una rotació i un estirament (escalat). A diferència del model *Neo-Hookean*, aquest té en compte les rotacions del gradient per a donar resultats més ajustats a les deformacions reals, i per tant extreure exactament l'estirament produït al material.

El model corrotacional fix és una modificació del model corrotacional per elasticitat lineal.

Aquest model es basa a usar la descomposició en valors singulars del gradient de deformació $\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, consultable a [A.4](#), on recordem $\mathbf{\Sigma}$ és una matriu diagonal, i anomenem σ_i als d elements d'aquesta diagonal, on d és la dimensió del problema.

La funció energètica del model corrotacional és la següent:

$$\Psi(\mathbf{F}) = \mu \sum_i^d (\sigma_i - 1)^2 + \frac{\lambda}{2} (J - 1)^2 \quad (2.59)$$

En aquest cas, és important veure que $J = \prod_i^d \sigma_i$, ja que expressa el grau de deformació sense les rotacions, i que és equivalent $\Psi(\mathbf{F}) = \Psi(\mathbf{\Sigma})$.

Per obtenir el nostre tensor \mathbf{P} hem de derivar aquesta equació [2.59](#) en funció del gradient de deformació \mathbf{F} , per fer això de manera clara hem d'expandir el primer terme de Ψ :

$$\mu \sum_i^d (\sigma_i - 1)^2 = \mu \left(\sum_i^d \sigma_i^2 - 2 \sum_i^d \sigma_i + d \right) \quad (2.60)$$

D'aquesta manera hem d'aconseguir derivar els dos sumatoris en funció de \mathbf{F} . La derivació d'aquests escalars en funció de matrius no serà vista en aquest document, consultar [\[7\]](#).

$$\frac{\partial}{\partial \mathbf{F}} \sum_i^d \sigma_i^2 = 2\mathbf{F} \quad (2.61)$$

$$\frac{\partial}{\partial \mathbf{F}} 2 \sum_i^d \sigma_i = 2\mathbf{R} \quad (2.62)$$

On $\mathbf{F} = \mathbf{R}\mathbf{S}$ descomposició polar de \mathbf{F} (operació consultable a [A.3](#)), que recordem podem calcular \mathbf{R} a través de la descomposició en valors singulars $\mathbf{R} = \mathbf{U}\mathbf{V}^T$.

El segon terme de l'equació [2.59](#) es pot derivar amb la fórmula de *Leibniz* per determinants, o la regla de diferenciació de determinants $\frac{\partial J}{\partial \mathbf{F}} = J\mathbf{F}^{-T}$, i la regla de la cadena.

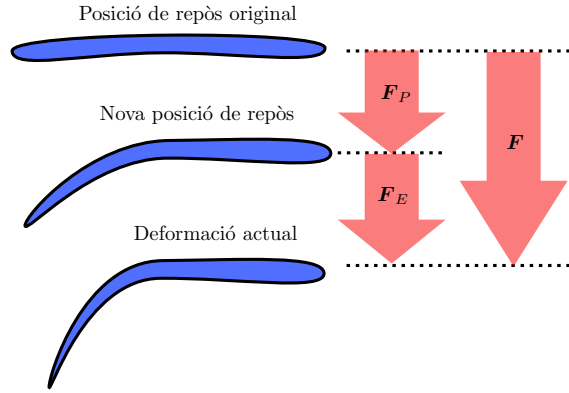


Figura 2.3: *Deformació plàstica*. Descomposició d'una deformació \mathbf{F} en plàstica no recuperable \mathbf{F}_P , i elàstica recuperable \mathbf{F}_E . (Elaboració pròpia)

Combinant aquests resultats obtenim el següent primer tensor de tensió de *Piola-Kirchhoff*:

$$\mathbf{P} = 2\mu(\mathbf{F} - \mathbf{R}) + \lambda(J - 1)J\mathbf{F}^{-T} \quad (2.63)$$

2.5.4. Plasticitat

Fins ara només hem tingut en compte deformacions elàstiques, de manera que s'utilitza tota la deformació per a computar la tensió que provocarà les forces de recuperació del material. Però així no actuen els sòlids reals, perquè no sempre són capaços de recuperar la forma original després d'una deformació. Això és la plasticitat.

La plasticitat és l'habilitat d'un material sòlid per sofrir una deformació, i provocar un canvi de forma no reversible sobre el sòlid. D'aquesta manera podem establir que qualsevol deformació d'un gradient de deformació \mathbf{F} pot ser factoritzada en una part elàstica \mathbf{F}_E i una part plàstica \mathbf{F}_P .

$$\mathbf{F} = \mathbf{F}_E \mathbf{F}_P \quad (2.64)$$

Un detall important a veure, és que tractem primer la deformació plàstica i després l'elàstica, ja que la recuperació ens tornarà a l'estat posterior de la deformació plàstica. Aquest fet es veu a la figura 2.3.

A causa d'aquesta factorització, per cada deformació establirem la posició resultant de la deformació plàstica com la nova posició de repòs.

Per marcar la plasticitat en un material, podem definir-la com el màxim de la component elàstica d'aquest. Això ho podem expressar amb els valors singulars σ_{Ei} de la descomposició de \mathbf{F}_E , obligant que la deformació expressada per aquests valors singulars sempre estigui dins d'un rang $[1 - \theta_c, 1 + \theta_s]$, per algunes constants θ_c i θ_s .

Com que la deformació plàstica no provocarà cap tensió, aquestes s'han de descartar del còmput dels respectius tensors, de manera que la derivació per obtenir el primer tensor de *Piola-Kirchhoff* de l'equació 2.51 serà només en funció de la part elàstica del gradient.

$$\mathbf{P} = \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}_E) \quad (2.65)$$

I per tant, per posar-ho en termes de tensió de *Cauchy*, podem reescriure l'equació 2.50 només tenint en compte la part elàstica de la deformació.

$$\boldsymbol{\sigma} = \frac{1}{J} \cdot \mathbf{P} \cdot \mathbf{F}_E^T = \frac{1}{J} \cdot \frac{\partial \Psi}{\partial \mathbf{F}_E} \cdot \mathbf{F}_E^T \quad (2.66)$$

En el cas concret de materials com la neu, aquests es tornaran més rígids sota una força de compressió. Aquest fenomen s'anomena enduriment, o *hardening* [5], i es pot controlar mitjançant els coeficients de *Lamé* μ i λ .

Veurem com aplicar aquesta plasticitat més endavant a la pràctica en la secció 4.4.

En el capítol d'elasto-plasticitat de Yuu [16], es pot trobar més informació sobre més aplicacions de la plasticitat.

3. Necessitats del simulador

A l'hora de dur a terme una simulació de qualsevol mena, hi ha nombrosos factors a tenir en compte per ajustar els resultats d'aquesta, sempre en funció dels requeriments per a l'execució de la simulació i del material a simular.

En aquest apartat es presenten els diversos punts a tenir en compte a l'hora de definir un sistema a simular, així com les possibles tècniques a emprar per a resoldre cada pas de la simulació com un problema dependent del pas anterior.

Cal afegir, que tot i que no és essencial per aquest treball, és convenient tenir una noció de les equacions de govern d'un sistema, i la seva forma dèbil, molt comuns en la literatura, i explicades a l'apèndix [A.8](#).

3.1. Discretització

Per tal d'avaluar de manera correcta i eficient els diferents models a simular, variables, i propietats dels materials, és necessari transformar components contínues a valors discrets, per poder usar certes estructures de dades, o simplement per a ser avaluables.

3.1.1. Espai *Eulerià*

Fins ara, tota la descripció de la deformació d'un material ha estat explicada com la transformació entre una multitud de coordenades de material *Lagrangianes* sense deformar, a les seves respectives posicions espacials *Eulerianes*. Per poder representar les posicions *Eulerianes*, és necessari discretitzar l'espai en petits volums que formen part d'una zona del medi continu.

Més concretament definirem tot l'espai on és possible que s'hi produeixin deformacions, i subdividirem aquest espai en petits volums que tractarem com coordenades *Eulerianes*.

Evidentment, en dues dimensions en comptes de subdividir en volums, es subdivideix en regions com s'observa a la figura [3.1](#). Aquesta estructura transforma les propietats a controlar a nivell de partícula a un sumatori a cada cel·la de l'espai *Eulerià*, el qual anomenarem graella, de cada partícula compatible.

La subdivisió ens permet ajustar en qualsevol moment la precisió de la simulació, ja que si la mida de les cel·les tendeix a zero, podem considerar subdivisions infinites, i per tant espai continu (dins d'una zona limitada).

D'aquesta manera, controlant les dades a cadascuna de les cel·les, i la interacció entre cel·les properes (per dissipació de líquid per exemple) podem simular *Euleriàment* un fluid o material.

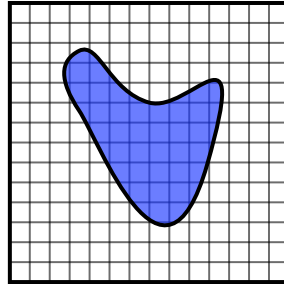


Figura 3.1: *Discretització de l'espai*. Limitació de l'espai a simular al rectangle negre exterior, subdividit en petites regions on es discretitzen les partícules del cos blau. (Elaboració pròpia)

3.1.2. Partícules *Lagrangianes*

En descripció material, o *Lagrangiana*, considerem les deformacions respecte de porcions infinitesimals del cos sense deformar. Com que és inviable controlar el material a escala de partícules moleculars, considerem partícules seccions del material sense deformar.

Cadascuna d'aquestes regions es defineixen per una massa, posició, volum, velocitat, i altres característiques concretes. De manera que aquestes partícules materials determinen el comportament de porcions estables del cos original. A diferència de la graella, l'espai on aquest material pot interactuar no està limitat, però resulta molt costós realitzar els còmputos si s'ha de tenir en consideració la interacció entre totes les partícules (complexitat com a mínim quadràtica respecte del nombre de partícules).

3.1.3. Temps

El temps és continu, però en les simulacions només ens interessa l'estat del sistema cada un cert interval de temps Δt .

De manera que donat el mateix cos Ω , podem denotar l'estat del mateix en un instant n mitjançant un superíndex Ω^n .

La diferència entre usar n i t per denotar instants de temps, és que l'últim és una variable continua, i és n discreta.

Per veure bé la diferència, podem denotar un següent interval de temps, quan $\Omega^n = \Omega^t$, podem escriure $\Omega^{n+1} = \Omega^{t+\Delta t}$.

Aquesta notació no només s'aplica al cos sencer Ω , sinó també a les diferents propietats de les partícules; com la velocitat v_p^n , en l'instant de temps n de la partícula p .

3.2. Interacció partícula-graella

Ja que la resolució del sistema, a cada instant de temps, usant únicament partícules és molt costosa, es van desenvolupar una sèrie de tècniques anomenades *Particle-in-cell* o PIC [17] les quals permeten aproximar la resolució de les equacions diferencials que defineixen les simulacions.

La idea darrere PIC és controlar un seguit de partícules en un espai continu, i paral·lelament en una graella *Euleriana* computar la informació que definirà el futur estat de les partícules, generant les interaccions necessàries. Generalment, és molt més senzill computar les forces de tensió en la graella, i després traslladar-les a les partícules.

3.2.1. Interpolació

Per a traspassar la informació de la partícula a la graella, i viceversa, es fan servir funcions *Eulerianes* d'interpolació. Denotem aquesta interpolació en la cel·la de la graella \vec{i} , on $\vec{i} = (i \ j \ k)$ posicions o índexs de la graella en 3 dimensions, que per simplicitat seran anomenats i , utilitzant la funció $N_i(\vec{x})$. Donada una partícula p i la seva posició en l'espai \vec{x}_p , la funció $N_i(\vec{x}_p) = w_{ip}$ associa per cada partícula p i cel·la i un pes w_{ip} , el qual determina la intensitat de la interacció entre l'anomenada partícula i cel·la. Normalment aquesta intensitat és inversament proporcional a la distància.

Per il·lustrar aquesta interpolació, podem formular la funció N_i com el producte d'una funció unidimensional N en cadascun dels eixos, a partir de la posició de la partícula p , $\vec{x}_p = (x_p \ y_p \ z_p)$, i la posició de la cel·la i , $x_i = (x_i \ y_i \ z_i)$:

$$N_i(\vec{x}_p) = N\left(\frac{1}{h}(x_p - x_i)\right) \cdot N\left(\frac{1}{h}(y_p - y_i)\right) \cdot N\left(\frac{1}{h}(z_p - z_i)\right) \quad (3.1)$$

On h marca la mida de cadascuna de les cel·les, i per tant la separació entre cel·les, de manera que N és independent de la mida de la graella. També suposem que les cel·les són quadrades, ja que en el cas contrari certes computacions es compliquen.

Ara el dilema és escollir la funció d'interpolació unidimensional N , tenint en compte el cost de la seva computació, amplada del filtre, i la suavitat del resultat reflectida en el gradient ∇w_{ip} .

A la pràctica, s'usen funcions *splines* quadràtiques o cúbiques, ja que són simples i senzilles de computar.

Per exemple es mostra el següent *kernel* quadràtic, extret de [7]:

$$N(x) = \begin{cases} \frac{3}{4} - |x|^2 & 0 \leq |x| < \frac{1}{2} \\ \frac{1}{2}(\frac{3}{2} - |x|)^2 & \frac{1}{2} \leq |x| < \frac{3}{2} \\ 0 & \frac{3}{2} \leq |x| \end{cases} \quad (3.2)$$

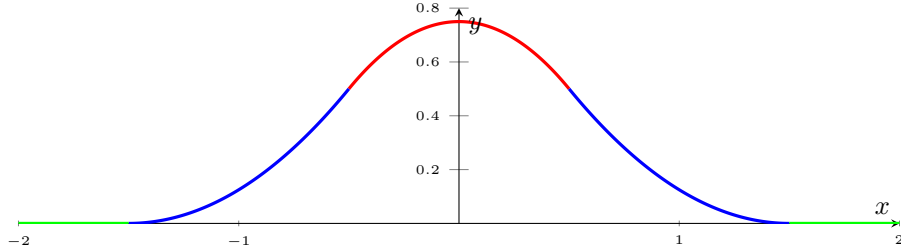


Figura 3.2: *Funció d'interpolació, spline quadràtica.* Gràfic de la *spline* de l'equació 3.2, on el primer terme $y = \frac{3}{4} - |x|^2$ està definit en vermell (domini $|x| < \frac{1}{2}$), el segon $y = \frac{1}{2}(\frac{3}{2} - |x|)^2$ en blau (domini $\frac{1}{2} \leq |x| < \frac{3}{2}$), i l'últim en verd $y = 0$ (domini $\frac{3}{2} \leq |x|$). (Elaboració pròpia)

El qual, per una partícula a una distància major de $\frac{1}{2}$ del centre d'una cel·la, la funció canvia i es comporta de manera més relaxada perquè es considera que la partícula ha entrat en l'àrea d'efecte d'una altra cel·la, fins a una distància de $\frac{3}{2}$ que ja és a dues cel·les de separació.

El següent gràfic 3.2 mostra aquesta funció per parts.

Amb aquesta funció d'interpolació podem tant emmagatzemar informació ponderada a les cel·les de la graella, com recuperar-la a posteriori usant la mateixa ponderació.

Emprant aquesta tècnica, cada partícula afectarà les corresponents cel·les que envolten la seva posició transformada i discretitzada. En dos dimensions 9 cel·les, i 27 en tres dimensions.

Per això, cada cel·la es veurà influenciada per totes les partícules fins a dues cel·les de separació, o a $2 \cdot h$ de distància, fet que es reflectirà en el moment de recuperar la informació tractada.

Aquest fet ens permet computar senzillament interaccions entre partícules properes.

3.2.2. PIC

Donada la funció d'interpolació N , les tècniques *particle-in-cell* ens permeten computar i interpolat la massa i el moment, per exemple, de la següent manera en un instant de temps n en cadascuna de les cel·les i de la graella.

$$m_i^n = \sum_p w_{ip}^n \cdot m_p \quad (3.3)$$

$$m_i^n \cdot \vec{v}_i^n = \sum_p w_{ip}^n \cdot m_p \cdot \vec{v}_p^n \quad (3.4)$$

On cal recordar que $w_{ip}^n = N_i(\vec{x}_p)$.

Amb les dades discretitzades a cadascuna de les cel·les, podem dur a terme certes computacions realitzant un processament de la graella.

Per exemple, és comú actualitzar la velocitat al següent instant de temps en el processament de la graella $\vec{v}_i^n \rightarrow \tilde{v}_i^{n+1}$. Notar que s'ha usat \tilde{v}_i^{n+1} , en comptes de \vec{v}_i^{n+1} , per diferenciar de la velocitat resultat de la interpolació a cada cel·la de la extrapolada a nivell de partícula en el següent instant de temps.

Amb aquesta nova velocitat calculada a cadascuna de les cel·les \tilde{v}_i^{n+1} , podem transmetre la informació altra vegada cap a les respectives partícules.

$$\vec{v}_p^{n+1} = \sum_i w_{ip}^n \cdot \tilde{v}_i^{n+1} \quad (3.5)$$

Aquest mètode es millorarà amb *APIC* [18], a la secció 4.2.

3.3. Evolució del gradient de deformació

A cada moment de temps, en què es modifiquen diferents porcions del material en funció de les forces que l'afecten (tant externes com internes), la deformació del mateix variarà i per tant el gradient de deformació s'ha de modificar de manera coherent.

És important notar que cada partícula tindrà el seu propi gradient de deformació.

Per actualitzar aquest gradient, podem definir la següent aproximació:

$$\mathbf{F}_p^{n+1} = \mathbf{F}_p^n + \Delta t \cdot \frac{\partial}{\partial t} \mathbf{F}_p^{n+1} \quad (3.6)$$

Ara bé, per calcular la derivada del gradient en funció de dos intervals de temps diferents, podem utilitzar l'equació de la derivada del gradient 2.12 ja vista en la secció 2.2, recordem $\frac{\partial}{\partial t} \mathbf{F} = \nabla \vec{v} \cdot \mathbf{F}$, però aplicant el següent:

$$\vec{V}(\vec{\mathbf{X}}, t + \Delta t) = \vec{v}(\phi(\vec{\mathbf{X}}, t), t + \Delta t) \quad (3.7)$$

Això és degut al fet que, com s'ha vist en l'apartat anterior 3.2.2, és possible calcular la velocitat *Lagrangiana* en el següent instant de temps en funció de la mateixa velocitat en l'instant de temps anterior, a través de la graella *Euleriana*; d'aquesta manera, la derivada queda:

$$\frac{\partial}{\partial t} \mathbf{F}_p^{n+1} = \nabla \vec{v}^{n+1}(\vec{x}_p^n) \cdot \mathbf{F}_p^n \quad (3.8)$$

És a dir, el gradient de la velocitat en l'instant $n + 1$ (calculat a través de la posició de la partícula en l'espai en l'instant n), multiplicat pel gradient de deformació previ.

Amb aquest terme, podem definir el nou gradient com a:

$$\mathbf{F}_p^{n+1} = \mathbf{F}_p^n + \Delta t \cdot \nabla \vec{v}^{n+1}(\vec{x}_p^n) \cdot \mathbf{F}_p^n = (\mathbf{I} + \Delta t \cdot \nabla \vec{v}^{n+1}(\vec{x}_p^n)) \cdot \mathbf{F}_p^n \quad (3.9)$$

Ara bé, encara no podem computar el gradient de la velocitat, tot i que sí que hem pogut computar la velocitat en l'apartat anterior mitjançant un sumatori ponderat a la graella:

$$\vec{v}_p^{n+1} = \sum_i w_{ip}^n \cdot \tilde{v}_i^{n+1} \quad (3.5 \text{ revisitada})$$

De manera que podem calcular el gradient, en forma de matriu, de la següent forma:

$$\nabla \vec{v}_p^{n+1} = \sum_i \tilde{v}_i^{n+1} \cdot (\nabla w_{ip}^n)^T \quad (3.10)$$

Ja que \tilde{v}_i^{n+1} és la dada de la graella que tractem com a constant, i transposem el gradient de la funció d'interpolació per obtenir aquesta matriu.

Per tant, obtenim una manera de computar en tot moment un nou gradient de deformació:

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \cdot \sum_i \tilde{v}_i^{n+1} \cdot (\nabla w_{ip}^n)^T) \cdot \mathbf{F}_p^n \quad (3.11)$$

El gradient de la funció d'interpolació w_{ip}^n és computable en qualsevol moment, ja que el tenim disponible a l'equació 3.2, per exemple.

4. *Material point method*

El *Material point method*, o *MPM*, és un mètode de simulació de comportaments d'objectes i cossos deformables, així com fluids, de manera híbrida.

Normalment els simuladors són o bé *Lagrangians*, i mantenen un sistema de partícules, o bé són *Eulerians* i mantenen una regió en forma de graella a on es produeix la simulació. Un sistema híbrid agafa, per dir-ho d'alguna manera, el millor dels dos mètodes en un de sol.

La idea és mantenir un sistema de partícules, però discretitzar tots els càlculs d'interacció en una graella, la qual s'utilitza només en la realització d'aquestes computacions i després de cada instant de temps s'esborra. De fet, ja hem vist com funciona aquesta interacció en la secció 3.2.2.

4.1. Mètode complet

En aquesta secció es descriu el mètode complet de *MPM*, referint a les seccions adequades per a cada pas.

En concret s'il·lustra la versió de *MLS-MPM* de Yuanming [8].

1. **Discretització de les dades de partícula a la graella.** També anomenada *particle to grid*, o *P2G*. Consisteix a traspassar totes les dades possibles a la graella, usant la funció d'interpolació N , definida a la secció 3.2.1.

En concret discretitzarem la massa de les partícules i el seu moment lineal, el qual es pot obtenir multiplicant la seva massa per la velocitat de la mateixa partícula. Per a transmetre aquest moment, és preferible usar *APIC*, explicat a la següent secció 4.2, i en concret utilitzant el seu vessant en *MLS* amb l'equació 4.16 per transmetre el moment tenint en compte altres forces internes.

2. **Actualització de la graella.** Processament de la graella per actualitzar el moment emmagatzemat i transformar a velocitat. En aquest punt s'ha d'utilitzar integració explícita o implícita per aplicar aquestes forces d'interacció.

Per actualitzar el moment amb *MLS*, s'ha d'usar l'equació 4.9.

Finalment, per transformar el moment lineal a velocitat, és tan senzill com dividir el moment per la massa emmagatzemada a cada cel·la i , $\tilde{v}_i^{n+1} = (m \cdot \tilde{v})_i^{n+1} / m_i^n$.

3. **Col·lisions externes en graella.** Usar la discretització de la graella per computar la interacció amb elements externs al sistema de partícules, com s'explica a la secció 4.5.

4. **Graella a partícules.** També anomenada *grid to particle*, o *G2P*. Consisteix en la recuperació de les dades de la graella a les partícules, així com la velocitat v_p^{n+1} amb l'equació 3.5, $\vec{v}_p^{n+1} = \sum_i w_{ip}^n \cdot \vec{v}_i^{n+1}$, juntament amb altres coeficients necessaris per *APIC* de l'equació 4.4, a veure en la següent secció 4.2.
5. **Actualització del gradient de deformació.** Actualitzar el gradient \mathbf{F} tal com s'ha vist a l'equació 3.11; o emprar l'aproximació que aporta *MLS* aprofitant els càlculs d'*APIC*, amb l'equació 4.8.
6. **Actualització de la plasticitat.** Aplicar la plasticitat (vista a la secció 2.5.4) al gradient de deformació corresponent, i actualitzar paràmetres de *Lamé* segons la secció 4.4.
7. **Col·lisions externes en partícules.** Aplicar sistemes de col·lisions per a computar col·lisions realistes, tot i que més costoses. Aquest punt no el veurem en aquest document. Consultar Stomakhin [5] per veure el mètode en si, i Ericson [19] per a una base en sistemes de col·lisions.
8. **Advecció de les partícules.** Computar noves posicions usant la velocitat calculada: $x_p^{n+1} = x_p^n + \Delta t \cdot v_p^{n+1}$.

4.2. APIC

El mètode *affine particle-in-cell* [18] és una millora de l'anterior *PIC* (vegeu secció 3.2.2), amb l'avantatge de què no és dissipatiu, i segueix mantenint una certa estabilitat.

Aquest es basa a tenir en compte el moment angular, extrapolant una aproximació a un tensor d'inèrcia (Consultar [10] per més informació sobre aquests tensors), imaginant que la modificació de la velocitat és una transformació afí respecte una posició fixa.

Aquesta transformació afí es representa amb la matriu \mathbf{C}_p^n , pròpia de cada partícula, identificant la contribució de velocitat en una cel·la i d'una graella:

$$\vec{v}_p^n + \mathbf{C}_p^n \cdot (\vec{x}_i - \vec{x}_p^n) \quad (4.1)$$

On \vec{x}_i identifica una posició, a distància constant, en cada cel·la de la graella i (usualment el centre). D'aquesta manera podem definir la transferència d'informació partícula-graella com al següent sumatori.

$$(m \cdot \vec{v})_i^n = \sum_p w_{ip}^n \cdot m_p \cdot (\vec{v}_p^n + \mathbf{C}_p^n \cdot (\vec{x}_i - \vec{x}_p^n)) \quad (4.2)$$

Per entendre que és exactament el tensor \mathbf{C}_p^n , introdueixo els tensors \mathbf{B}_p^n i \mathbf{D}_p^n amb la següent relació.

$$\mathbf{B}_p^n = \mathbf{C}_p^n \cdot \mathbf{D}_p^n \quad (4.3)$$

On es defineix el tensor \mathbf{B}_p^n en funció de la velocitat entre transferències i la posició de la partícula en l'instant anterior. Recordar la notació explicada a la

secció 3.2.2 sobre \hat{v}_i .

$$\mathbf{B}_p^{n+1} = \sum_i w_{ip}^n \cdot \tilde{v}_i^{n+1} \cdot (\vec{x}_i - \vec{x}_p^n)^T \quad (4.4)$$

$$\mathbf{D}_p^n = \sum_i w_{ip}^n \cdot (\vec{x}_i - \vec{x}_p^n) \cdot (\vec{x}_i - \vec{x}_p^n)^T \quad (4.5)$$

Davant d'aquestes dues definicions, podem veure com el tensor $\mathbf{C}_p^n = \mathbf{B}_p^n \cdot (\mathbf{D}_p^n)^{-1}$ produeix una correcció a la velocitat de la partícula, en funció de la nova posició d'aquesta, d'acord amb la velocitat computada en les cel·les corresponents a la partícula en l'instant de temps anterior i la posició prèvia d'aquesta.

Per tant, és més exacte escriure l'equació 4.2 com:

$$(m \cdot \tilde{v})_i^n = \sum_p w_{ip}^n \cdot m_p \cdot (\vec{v}_p^n + \mathbf{B}_p^n \cdot (\mathbf{D}_p^n)^{-1} \cdot (\vec{x}_i - \vec{x}_p^n)) \quad (4.6)$$

Convenientment, el tensor \mathbf{D}_p^n té una forma molt senzilla en el cas d'usar funcions d'interpolació N quadràtiques $\mathbf{D}_p^n = \frac{1}{4} \Delta x^2 \mathbf{I}$, on Δx és la mida de cadascuna de les cel·les de la graella.

4.3. *MLS*

El mètode de *moving least squares*, o *MLS*, és un mètode per a reconstruir funcions contínues a partir d'un set de mostres en l'espai, donant lloc a una aproximació d'aquesta funció a través de la minimització de l'error ponderat.

Aquest mètode ha estat aplicat per Hu et al. a [8] per discretitzar la forma dèbil de les equacions generals del sistema, en l'equació A.97. Les implicacions d'aquest fet les trobem en aquest apartat.

4.3.1. Actualització del gradient de deformació

L'actualització del gradient de deformació \mathbf{F} ha estat definida *Lagrangianament* en la secció 3.3 de la següent manera.

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \cdot \nabla \vec{v}^{n+1}(\vec{x}_p)) \cdot \mathbf{F}_p^n \quad (3.9 \text{ revisitada})$$

On, segons *MPM* tradicional, es defineix el gradient de la velocitat $\nabla \vec{v}_p^{n+1} = \sum_i \tilde{v}_i^{n+1} \cdot (\nabla w_{ip}^n)^T$, com mostrat a l'equació 3.10.

Per altra banda, com que *MLS* discretitza i subdivideix el domini continu, reflectint en l'equació dèbil del sistema A.97, es poden aproximar equacions contínues amb mostres a les cel·les *Eulerianes*.

Per això es considera que el tensor \mathbf{C}_p d'*APIC*, de l'apartat anterior 4.2, és equivalent a aquesta derivada.

$$\nabla \vec{v}_p^{n+1} = \mathbf{C}_p^{n+1} \quad (4.7)$$

Això és degut al fet que la construcció d'aquest tensor és el sumatori de mostres ponderades de velocitat, en funció de les variacions de les components, la qual cosa gràcies a l'aproximació de *MLS* ara podem considerar la derivada. D'aquesta manera podem recalculer el gradient de deformació \mathbf{F}_p a partir del tensor \mathbf{C}_p , que es pot aprofitar del còmput anterior d'*APIC*.

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \cdot \mathbf{C}_p^{n+1}) \cdot \mathbf{F}_p^n \quad (4.8)$$

4.3.2. Actualització de la graella

El moment emmagatzemat a la graella s'ha d'actualitzar, i en el cas de *MLS* podem tenir en compte les forces hiperelàstiques d'una manera particular, com mostren les següents seccions.

Suposant que hem sigut capaços de discretitzar totes aquestes dades de les partícules fins a la cel·la i de la graella, podem considerar el nou moment com:

$$(m \cdot \tilde{v})_i^{n+1} = (m \cdot \tilde{v})_i^n + \Delta t \cdot (m_i^n \cdot g + \vec{f}_i^n) \quad (4.9)$$

On g és la gravetat, i \vec{f}_i^n les forces hiperelàstiques. D'aquesta manera podem, duent a terme una única passada per la graella, actualitzar-ne els moments lineals.

Posteriorment s'han de transformar els moments lineals a velocitat, simplement dividint per la massa de la cel·la.

$$\tilde{v}_i^{n+1} = (m \cdot \tilde{v})_i^{n+1} / m_i^n \quad (4.10)$$

4.3.3. Integració explícita de forces

Durant una deformació, la força de retorn del cos es pot trobar en funció de l'energia potencial, de manera que es pot derivar la força afectant el model. Recordem que podem calcular la força \vec{f} en funció d'una posició \vec{x} i de una funció escalar E corresponent a l'energia potencial, $\vec{f}(\vec{x}) = -\frac{\partial E}{\partial \vec{x}}$.

D'acord amb l'observat a la secció 2.5.1, podem definir l'energia d'una partícula p com la seva densitat energètica Ψ_p (a partir del seu gradient de deformació) pel seu volum V_p^0 , el qual és constant, amb la següent funció $E_p^n = V_p^0 \cdot \Psi_p(\mathbf{F}_p^n)$. Per tant, l'energia total del sòlid a simular es descriu pel seu sumatori d'energies:

$$E^n = \sum_p V_p^0 \cdot \Psi_p(\mathbf{F}_p^n) \quad (4.11)$$

Tanmateix, per obtenir les forces s'ha de derivar l'anterior expressió en funció de la posició en l'espai. A més a més, podem discretitzar aquestes forces segons cada cel·la i de la graella, de manera que la diferenciació és fa segons la distància entre la partícula i la cel·la $(\vec{x}_i - \vec{x}_p^n)$:

$$\vec{f}_i^n = -\frac{\partial E}{\partial \vec{x}_i} = -\sum_p N_i(\vec{x}_p^n) \cdot V_p^0 \cdot \mathbf{D}_p^{-1} \cdot \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}_p) \cdot \mathbf{F}_p^{nT} \cdot (\vec{x}_i - \vec{x}_p^n) \quad (4.12)$$

On \mathbf{D}_p és el tensor en d'APIC (secció 4.2), i $\frac{\partial \Psi}{\mathbf{F}}$ és, segons l'equació 2.51, el primer tensor de tensió de Piola-Kirchhoff \mathbf{P} (secció 2.4.3).

Aquesta equació 4.12 ens permet calcular les forces de manera explícita en cada cel·la del sistema, en la secció 4.3.4 es veurà com utilitzar aquesta expressió.

4.3.4. Partícula a graella

Donada la força \vec{f}_p^n que afecta una partícula p , la seva massa m_p i l'interval de temps emprat en la simulació Δt , en podem computar una velocitat corresponent a la dispersió de forces.

$$\vec{v}_p^n = \Delta t \cdot m_p^{-1} \cdot \vec{f}_p^n \quad (4.13)$$

I per tant, segons l'equació anterior 4.12, podem completar aquesta velocitat com:

$$\vec{v}_p^n = -\Delta t \cdot m_p^{-1} \cdot V_p^0 \cdot \mathbf{D}_p^{-1} \cdot \frac{\partial \Psi}{\mathbf{F}}(\mathbf{F}_p) \cdot \mathbf{F}_p^{nT} \cdot (\vec{x}_i - \vec{x}_p^n) \quad (4.14)$$

Aquest fet el podem aplicar per resoldre, satisfactòriament, l'equació d'APIC 4.2, la qual recordem:

$$(m \cdot \vec{v})_i^n = \sum_p w_{ip}^n \cdot m_p \cdot (\vec{v}_p^n + \mathbf{C}_p^n \cdot (\vec{x}_i - \vec{x}_p^n)) \quad (4.2 \text{ revisitada})$$

Donada aquesta informació, és més senzill de veure el còmput factoritzant l'equació amb un tensor \mathbf{Q}_p corresponent a:

$$\mathbf{Q}_p^n = -\Delta t \cdot V_p^0 \cdot \mathbf{D}_p^{-1} \cdot \frac{\partial \Psi}{\mathbf{F}}(\mathbf{F}_p) \cdot \mathbf{F}_p^{nT} + m_p \cdot \mathbf{C}_p^n \quad (4.15)$$

De manera que el pas de partícula a graella ($P2G$), derivat de l'equació d'APIC 4.2 i amb , es pot dur a terme amb aquest tensor \mathbf{Q}_p .

$$(m \cdot \vec{v})_i^n = \sum_p w_{ip}^n \cdot ((m_p \cdot \vec{v})_p^n + \mathbf{Q}_p^n \cdot (\vec{x}_i - \vec{x}_p^n)) \quad (4.16)$$

4.4. Plasticitat

Tal com s'ha explicat a la secció 2.5.4 sobre plasticitat, podem controlar aquest fenomen factoritzant el gradient de deformació \mathbf{F} en les seves components elàstiques i plàstiques $\mathbf{F} = \mathbf{F}_E \mathbf{F}_P$, controlant que els valors singulars de \mathbf{F}_E estiguin dins d'un rang $[1 - \theta_c, 1 + \theta_s]$, per algunes constants θ_c i θ_s , i utilitzar només aquest gradient de deformació elàstic per als còmput de tensió.

Per a tractar el com resoldre aquest problema, suposarem que podem escriure la factorització sense aplicar plasticitat com a:

$$\mathbf{F}^{n+1} = \tilde{\mathbf{F}}_E^{n+1} \mathbf{F}_P^n \quad (4.17)$$

De manera que s'ha d'extreure la plasticitat del tensor \mathbf{F}_E^{n+1} per a construir el gradient de plasticitat del següent moment de temps \mathbf{F}_P^{n+1} . És important veure que de totes maneres el resultat serà el mateix gradient de deformació \mathbf{F}^{n+1} . Per a eliminar aquesta plasticitat de $\tilde{\mathbf{F}}_E^{n+1}$ s'ha de realitzar la *singular value decomposition* (operació consultable a [A.4](#)):

$$\tilde{\mathbf{F}}_E^{n+1} = \mathbf{U}_E^{n+1} \tilde{\Sigma}_E^{n+1} \mathbf{V}_E^{n+1^T} \quad (4.18)$$

D'on podem computar uns nous valors singulars Σ , aplicant el rang esmentat anteriorment, $[1 - \theta_c, 1 + \theta_s]$, sobre tots els valors singulars de $\tilde{\mathbf{F}}_E^{n+1}$. Construint un nou gradient de deformació elàstica:

$$\mathbf{F}_E^{n+1} = \mathbf{U}_E^{n+1} \Sigma_E^{n+1} \mathbf{V}_E^{n+1^T} \quad (4.19)$$

Per tant podem redefinir el mateix gradient de deformació de l'equació [4.17](#):

$$\mathbf{F}^{n+1} = \mathbf{F}_E^{n+1} \mathbf{F}_P^{n+1} \quad (4.20)$$

On el gradient de plasticitat \mathbf{F}_P^{n+1} n'és una incògnita, fàcilment resoluble aïllant-la:

$$\mathbf{F}_P^{n+1} = (\mathbf{F}_E^{n+1})^{-1} \mathbf{F}^{n+1} \quad (4.21)$$

Per a controlar l'enduriment del material, com s'ha mencionat a l'apartat [2.5.4](#), cal modificar els coeficients de *Lamé* μ i λ en funció de la compressió plàstica del material, de manera que aquests incrementin sota compressió i augmentin amb l'extensió.

$$\mu(\mathbf{F}_P) = \mu_o \cdot e^{\mathcal{E} \cdot (1 - J_P)} \quad (4.22)$$

$$\lambda(\mathbf{F}_P) = \lambda_0 \cdot e^{\mathcal{E} \cdot (1 - J_P)} \quad (4.23)$$

On J_P és el determinant de \mathbf{F}_P , i \mathcal{E} el coeficient de duresa, el qual acostuma a prendre valors entre 3 i 10.

Aquest fet influirà en el còmput dels models, tant *Neo-Hookean*, com corrotacional fix, de la secció [2.5](#).

Un fet interessant a veure, és que només necessitem el gradient de deformació de plasticitat per a calcular els nous paràmetres de *Lamé*, ja que s'emprarà el gradient d'elasticitat \mathbf{F}_E pels càlculs de tensió.

D'aquesta forma, resulta més còmode només emmagatzemar l'anomenat tensor \mathbf{F}_E i el determinant J_P a cada partícula.

Aquest determinant J_P el podem calcular seguint l'equació [4.21](#) i [4.17](#), però amb determinants:

$$J_P^{n+1} = J_E^{n+1} / J^{n+1} \quad (4.24)$$

$$J_P^{n+1} = (\tilde{J}_E^{n+1} \cdot J_P^n) / J^{n+1} \quad (4.25)$$

On $\tilde{J}_E^{n+1} = \det(\tilde{\mathbf{F}}_E^{n+1})$, el determinant del gradient elàstic sense aplicar la plasticitat.

Podem limitar els valors d'aquest determinant J_P^{n+1} , limitant també les capacitats de deformació plàstica de l'objecte, provocant el trencament del mateix quan la deformació no sigui tractable..

4.5. Col·lisions externes en graella

Qualsevol cos rígid extern pot ser considerat a l'hora de dur a terme una simulació, ja sigui un cos immòbil com en moviment.

Un exemple comú d'aquest tipus de col·lisió és el mateix terreny en el què el fluid circula.

Per a tractar les col·lisions, hem de calcular les velocitats relatives amb la cel·la de referència.

$$\vec{v}_{i,\text{rel}} = \vec{v}_i - \vec{v}_{i,\text{obj}} \quad (4.26)$$

On \vec{v}_i és la velocitat a la cel·la i de la graella, i $\vec{v}_{i,\text{obj}}$ la velocitat del cos extern discretitzat a la cel·la i . D'aquesta manera obtenim la velocitat relativa al cos $\vec{v}_{i,\text{rel}}$.

En cas de col·lisions amb objectes immòbils, $\vec{v}_{i,\text{rel}} = \vec{v}_i$.

És important veure que només s'hauran d'aplicar col·lisions si el cos simulat i el cos a col·lidir no s'estan separant, fet que es pot calcular donada la normal $\hat{n}_{i,\text{obj}}$ de l'objecte discretitzada a la cel·la i . D'aquesta manera les col·lisions s'apliquen si $v_{i,n} = \vec{v}_{i,\text{rel}} \cdot \hat{n}_{i,\text{obj}} < 0$, on $v_{i,n}$ és un escalar corresponent a la velocitat normal a l'objecte de la cel·la i .

Només les velocitats tangencials al cos tindran un efecte, ja que les normals es veuran anul·lades per la col·lisió. Podem calcular les velocitats tangencials $\vec{v}_{i,t}$:

$$\vec{v}_{i,t} = \vec{v}_{i,\text{rel}} - \hat{n}_{i,\text{obj}} \cdot v_{i,n} \quad (4.27)$$

I podem calcular la nova velocitat relativa $\vec{v}'_{i,\text{rel}}$ segons un coeficient de fricció μ , aplicant fricció dinàmica:

$$\vec{v}'_{i,\text{rel}} = \vec{v}_{i,t} + \mu \cdot v_{i,n} \cdot \frac{\vec{v}_{i,t}}{|\vec{v}_{i,t}|} \quad (4.28)$$

També es podria tenir en compte un impuls viscos quan $|\vec{v}_{i,t}| \leq -\mu \cdot v_{i,n}$, anul·lant la velocitat relativa final $\vec{v}'_{i,\text{rel}} = 0$.

De manera que la velocitat final a la cel·la i és:

$$\vec{v}'_i = \vec{v}'_{i,\text{rel}} + \vec{v}_{i,\text{obj}} \quad (4.29)$$

5. Implementació *MPM*

En aquest apartat s'explicarà, en detall, la implementació realitzada d'un simulador de *MPM-MLS* en 3 dimensions, tal com l'explicat en el capítol anterior 4, de manera que permet calcular l'estat d'un sistema de deformacions després d'un temps Δt .

També es troben disponibles, en la següent carpeta de [Google Drive](#) i explicades en l'apèndix B, diverses animacions per il·lustrar les capacitats d'aquest simulador.

S'ometran tots els detalls de la prèvia implementació en dues dimensions, tot i que aquesta es troba disponible en els arxius adjunts a aquesta memòria, sota el projecte 2D.SIM. Una animació es troba disponible per mostrar les capacitats d'aquest projecte previ, amb nom *2d.gif* (Apèndix B.1).

La implementació del simulador en tres dimensions es troba en el projecte *Simulator3D*, concretament en els arxius *Simulator_3D.h* i *Simulator_3D.cpp*, també adjunts en l'apèndix C.2 d'aquesta memòria.

Tots els exemples mostrats són extraccions del codi d'aquests arxius, però sense certes etiquetes, espais de noms, i qualificadors, en pro de la facilitat de lectura. S'ha realitzat en el llenguatge de C++, concretament en la seva versió C++14.

5.1. Preàmbul

Abans d'entrar en l'algorisme principal de *MPM* cal introduir certes dependències, per evitar haver de programar funcionalitats complexes ja ben implementades, i descriure les estructures de dades a usar i la manera d'interactuar amb aquestes. També cal dir que tots els càlculs es duren a terme amb `floats`, a causa a la seva rapidesa versus els `double`.

Per a millorar la precisió de la simulació, l'espai en què aquesta es durà a terme serà un cub de dimensions $1 \times 1 \times 1$, de manera que totes les dades es trobaran en aquest rang.

Això és important, ja que ens obligarà a dur a terme conversions per discretitzar les posicions \vec{x}_p de les partícules p a la graella, de la següent manera.

$$\vec{i}_p = \lfloor \vec{x}_p \cdot \text{grid_size} \rfloor \quad (5.1)$$

On \vec{i}_p és l'índex de la graella de la partícula p , discretitzada segons la mida de cada dimensió de la graella `grid_size`.

5.1.1. Dependències

- **Eigen:** Una llibreria d'àlgebra lineal en C++ basada en *templates*, que inclou suport tant per matrius com per vectors, i evidentment algorismes com la *singular value decomposition* (Apartat A.4). Aquesta llibreria marcarà els tipus més bàsics interns, i és amb la que s'efectuaran tots els càlculs en l'algorisme de MPM. En concret s'usarà la revisió d'Eigen 3.3.7 [21].
- **GLM:** La llibreria de matemàtiques d'*OpenGL* és una de les opcions més comunes en projectes que usen vectors i matrius. D'aquesta manera l'input i output de dades es faran, en la mesura del possible, en els tipus d'aquesta llibreria. Particularment, s'utilitzarà la seva versió GLM 0.9.9 [22].

5.1.2. Estructures de dades

L'algorisme principal empra dues estructures de dades, les quals gestionen l'aspecte *Lagrangia* i *Euleria* de la simulació, respectivament. Totes aquestes estructures es troben declarades en l'arxiu `Simulator_3D.h`.

- **Partícules:** Les partícules seran emmagatzemades en un vector, amb tantes posicions com partícules al sistema, de manera que es poden afegir partícules en mig d'una simulació. Cadascuna de les partícules correspon a la següent informació.

```
Eigen::Array3f pos; // posició
Eigen::Array3f v;   // velocitat
Eigen::Matrix3f F;  // gradient de deformació
Eigen::Matrix3f C;  // APIC
float Jp;           // determinat de F (plasticitat)
```

On `Eigen::Matrix3f` són matrius quadrades de 3×3 .

Altres dades, com la massa `m`, el volum `vol`, paràmetres de Lamé i la duresa, s'emmagatzemen com a constants de la simulació, fins a introduir materials (secció 5.4).

- **Graella:** La graella, el vessant *Euleria* de la simulació, consisteix en una discretització de l'espai 3D en vòxels, com si es tractés d'un *octree*. Per simplificació s'emprarà un *array* de vectors d'Eigen de 4 posicions, de manera que hi hagi `grid_size`³ vectors en l'*array*, on `grid_size` és el nombre de subdivisions per dimensió de l'espai. Les components d'aquest vector corresponen al moment lineal o velocitat de la cel·la, depenent del pas de l'algorisme, en les 3 primeres components, i finalment la massa de la cel·la en la quarta. Més concretament `grid[i][j][k]=(v.x, v.y, v.z, m)`, en la cel·la identificada per `i`, `j`, `k`. És important veure que cada element d'aquesta graella correspon a 16 bytes (4 bytes per component), de manera que les dades cabran senceres en la mateixa línia de memòria *cache*, fet que ens estalviarà sincronitzacions, i millorarà el paral·lelisme.

- **Graella de físiques:** Graella empleada per emmagatzemar dades de les col·lisions, en aquest cas la normal de l'objecte a col·lidir discretitzada a la cel·la.
És un array amb el mateix nombre d'elements (cel·les) que l'anterior graella, però amb vectors de 3 posicions.

5.2. Algorisme

Seguint el mètode explicat a la secció 4.1, podem separar l'algorisme en 3 parts: *particle-to-grid* (*P2G*), processament de la graella, i *grid-to-particle* (*G2P*); juntament amb la inicialització del sistema i el reinici de la graella a cada iteració. Tots els passos de configuració de components de les partícules es realitzaran en el pas de *G2P*.

Per facilitar l'entesa de l'algorisme, s'adjunta l'esquelet d'aquest:

```
void step(float dt){
    // 0 - Reinici de la graella
    std::memset(grid,0,sizeofGrid);
    // 1 - P2G
    for(particle& p : this->particles){
        // 1.1: Calcular pesos per interpolar
        // la regió de 3x3x3 cel·les al voltant
        // de la posició de la partícula
        // 2.2: Calcular tensió
        // 2.3: Transferir moment
        for(cell& i : p.neighbourhood){
            // interpolat moment i massa
        }
    }
    // 2 - Càlcul de les velocitats a graella
    for(cell& i : this->grid){
        // 2.1: Convertir moment en velocitat
        // 2.2: Forçar límits de l'espai
        // 2.3: Aplicar físiques
    }
    // 3 - G2P
    for(particle& p : this->particles){
        // 4.1: Calcular pesos, igual que 1.1
        // 4.2: Calcular nova velocitat
        for(cell& i : p.neighbourhood){
            // 4.2.1: Interpolar velocitats
            // 4.2.2: Calcular C APIC
        }
        // 4.3: Advecció de la partícula
        // 4.4: Actualitzar gradient de deformació
        // 4.5: Aplicar plasticitat
    }
}
```

5.2.1. Inicialització

Primerament s'han d'introduir totes les partícules al sistema, i alhora inicialitzar el gradient de deformació i el tensor \mathbf{C} d'*APIC*.

Cal recordar que totes les posicions inicials han d'estar en el rang $(0, 1)$, per millorar la precisió i mantenir una invariant en l'algorisme. Corresponentment es pot afegir una velocitat inicial a les partícules, generalment la mateixa per a tot el mateix cos.

El gradient de deformació \mathbf{F} i el tensor \mathbf{C} s'inicien amb la matriu identitat \mathbf{I} , ja que no s'ha produït cap deformació encara. Igualment el determinant de la component plàstica J_p serà 1.

També s'han de computar els paràmetres de Lamé, comuns per cada material de la següent manera:

$$\lambda = \frac{E \cdot \nu}{(1 + \nu) \cdot (1 - 2 \cdot \nu)} \quad (2.57 \text{ revisitada})$$

$$\mu = \frac{E}{2 \cdot (1 + \nu)} \quad (2.58 \text{ revisitada})$$

5.2.2. *P2G*

Abans d'iniciar aquest pas s'ha de reiniciar tota la graella amb zeros, ja que funciona com una ubicació temporal on efectuar els càlculs. Podem usar la mateixa instrucció de C++ `std::memset`.

Posteriorment s'itera per totes les partícules, discretitzant les seves dades a la graella.

Discretització

Usant la funció d'interpolació unidimensional quadràtica N , de la secció 3.2.1, recordem:

$$N(x) = \begin{cases} \frac{3}{4} - |x|^2 & 0 \leq |x| < \frac{1}{2} \\ \frac{1}{2}(\frac{3}{2} - |x|)^2 & \frac{1}{2} \leq |x| < \frac{3}{2} \\ 0 & \frac{3}{2} \leq |x| \end{cases} \quad (3.2 \text{ revisitada})$$

Com que únicament s'obtiniran valors majors a zero en la cel·la on pertany la partícula, i en les dues cel·les adjacents, només tenint en compte una dimensió, podem ja definir els 3 pesos resultants com un vector de tres posicions,

corresponents a la cel·la anterior, actual, i següent.

$$\vec{N}(dx) = \begin{pmatrix} 0.5 \cdot (0.5 - dx)^2 \\ 0.75 - dx^2 \\ 0.5 \cdot (dx + 0.5)^2 \end{pmatrix} \quad (5.2)$$

On hem considerat que la distància entre cel·les consecutives és d'una unitat. Veure que dx en aquest cas és la distància des del centre de la cel·la a la partícula.

Com que aquesta interpolació s'ha de realitzar en les tres dimensions, podem tractar la interpolació en funció del vector posició directament i obtenir totes les dades en una matriu.

$$\mathbf{N}(\vec{dx}) = \begin{pmatrix} 0.5 \cdot (0.5 - \vec{dx})^2 \\ 0.75 - \vec{dx}^2 \\ 0.5 \cdot (\vec{dx} + 0.5)^2 \end{pmatrix} \quad (5.3)$$

D'aquesta manera, cada fila de \mathbf{N} indica una cel·la, i cada columna indica el pes a la corresponent coordenada.

Per calcular aquesta distància \vec{dx} , es diferencia la posició transformada de la partícula del centre de la cel·la en qüestió, calculat com l'índex de la graella més 0.5 unitats.

El codi C++ és el següent, donada la partícula p :

```
Array3f cell_if = p.pos * grid_size;
Array3i cell_i = cell_if.cast<int>(); // floor
// vector centre cel·la -> partícula
Array3f dx = (p.pos * grid_size) - (cell_i.cast<float>() + 0.5f);

// funció d'interpolació N
Array3f weights[3] = {
    0.5f * (0.5f - dx).square(),
    0.75f - (dx).square(),
    0.5f * (dx + 0.5f).square()
};
```

El pes per interpolat final és expressat per la multiplicació dels pesos en cadascuna de les components de la posició, recordem:

$$N_i(\vec{x}_p) = N\left(\frac{1}{h}(x_p - x_i)\right) \cdot N\left(\frac{1}{h}(y_p - y_i)\right) \cdot N\left(\frac{1}{h}(z_p - z_i)\right) \quad (3.1 \text{ revisitada})$$

Matriu afi \mathbf{Q}

Recordant l'equació 4.16, podem afegir les forces internes a l'etapa de $P2G$ computant el tensor \mathbf{Q}_p^n per cadascuna partícula.

Aquest, recordem que es defineix:

$$\mathbf{Q}_p^n = -\Delta t \cdot V_p^0 \cdot \mathbf{D}_p^{-1} \cdot \frac{\partial \Psi}{\mathbf{F}}(\mathbf{F}_p) \cdot \mathbf{F}_p^{nT} + m_p \cdot \mathbf{C}_p^n \quad (4.15 \text{ revisitada})$$

De manera que variarà en funció del model energètic usat $\frac{\partial \Psi}{\mathbf{F}}(\mathbf{F}_p)$, que recordem és equivalent al primer tensor de *Piola-Kirchhoff* $\mathbf{P}(\mathbf{F}_p)$ (equació 2.51).

Qualsevol tensor \mathbf{P} requereix els paràmetres de Lamé, potser modificats per la plasticitat, computats amb les equacions de la secció 4.4:

$$\mu(J_P) = \mu_o \cdot e^{\mathcal{E} \cdot (1 - J_P)} \quad (4.22 \text{ revisitada})$$

$$\lambda(J_P) = \lambda_o \cdot e^{\mathcal{E} \cdot (1 - J_P)} \quad (4.23 \text{ revisitada})$$

I en codi C++:

```
float e = std::exp(hardening * (1.0f - p.Jp));
float mu = mu * e;
float lambda = lambda * e;
```

Per altra banda, el tensor de tensió \mathbf{P} canviarà depenent de si usem el model *Neo-Hookean* (secció 2.5.2) o el model corrotacional fix (secció 2.5.3), vistos en aquest document.

En el cas del model *Neo-Hookean*, el tensor $\mathbf{P}_{\text{neo}}(\mathbf{F})$ es defineix com:

$$\mathbf{P}_{\text{neo}}(\mathbf{F}) = \mu(\mathbf{F} - \mathbf{F}^{-T}) + \lambda \lg(J) \mathbf{F}^{-T} \quad (2.53 \text{ revisitada})$$

I substituint a l'equació anterior 4.15, obtenim \mathbf{Q}_{neo} :

$$\mathbf{Q}_{p,\text{neo}}^n = -\Delta t \cdot V_p^0 \cdot \mathbf{D}_p^{-1} \cdot (\mu(\mathbf{F}^n \mathbf{F}_p^{nT} - \mathbf{I}) + \lambda \lg(J^n) \mathbf{I}) + m_p \cdot \mathbf{C}_p^n \quad (5.4)$$

També, podem usar el model corrotacional fix, amb el corresponent tensor $\mathbf{P}_{\text{cor}}(\mathbf{F})$ definit com:

$$\mathbf{P}_{\text{cor}}(\mathbf{F}) = 2\mu(\mathbf{F} - \mathbf{R}) + \lambda(J - 1)J\mathbf{F}^{-T} \quad (2.63 \text{ revisitada})$$

On \mathbf{R} és la part rotacional de la descomposició polar de \mathbf{F} (operació consultable a A.3), que podem calcular usant la *singular value decomposition* de \mathbf{F} , $\mathbf{F} = \mathbf{U}\Sigma\mathbf{V}^*$, amb l'equació A.34, $\mathbf{R} = \mathbf{U}\mathbf{V}^*$

Substituint també a l'equació anterior 4.15:

$$\mathbf{Q}_{p,\text{cor}}^n = -\Delta t \cdot V_p^0 \cdot \mathbf{D}_p^{-1} \cdot (2\mu(\mathbf{F}_p^n - \mathbf{R}_p^n) \mathbf{F}_p^{nT} + \lambda(J - 1)J\mathbf{I}) + m_p \cdot \mathbf{C}_p^n \quad (5.5)$$

És interessant veure com en substituir el tensor de tensió, tant en el model *Neo-Hookean* com corrotacional, el terme \mathbf{F}^{-T} ha estat eliminat, fet que es veurà reflectit en el rendiment del simulador ja que calcular una inversa d'una matriu és una operació molt costosa.

També recordar que el tensor $(\mathbf{D}_p)^{-1}$ és una matriu diagonal, que podem considerar un escalar (secció 4.2), concretament $\mathbf{D}_p^n = \frac{1}{4}\Delta x^2 \mathbf{I}$ en el nostre cas.

En codi C++, el tensor *Neo-Hookean* $\mathbf{Q}_{p,\text{neo}}^n$ es computa de la següent manera, on `affine` identifica aquest tensor.

```
// P Neo-Hookean multiplicat per F^T
Matrix3f PF_t = mu *
    ((p.F * (p.F).transpose() - Matrix3f::Identity()))
    + (Matrix3f::Identity() * (lambda * std::log(J)));
// D^-1
float Dinv = (4.0f * grid_size * grid_size);

Matrix3f stress = (-dt * volume * Dinv) * PF_t;
// Q
Matrix3f affine = stress + mass * p.C;
```

De manera idèntica podem trobar $\mathbf{Q}_{p,cor}^n$, computant la tensió corrotacional $\mathbf{P}_{cor}(\mathbf{F})$, i usant-la per computar la matriu afí anterior, duent a terme:

```
// Calcular U i V de la svd
JacobiSVD<Matrix3f, NoQRPreconditioner>
    svd(p.F, Eigen::ComputeFullU | Eigen::ComputeFullV);
// Construir R=UV^*, on V^*=V^T=V^-1
Matrix3f R = svd.matrixU() * svd.matrixV().transpose();
// P corrotacional multiplicat per F^T
Matrix3f PF_t = 2.0f * mu * (p.F - R) * (p.F).transpose()
    + (Matrix3f::Identity() * (lambda * (J - 1.0f) * J))
```

Sense tenir en compte quin dels dos models usem, obtenim un tensor \mathbf{Q}_p^n adient per a discretitzar les dades en la graella. Codi complet a l'apèndix C.2.1.

Transferència del moment

La fase culminant del *P2G* consisteix a transferir el moment de totes les partícules a cadascuna de les cel·les de la graella, tal com a la secció 4.3.4 s'ha indicat, d'acord amb la transferència d'*APIC*:

$$(m \cdot \tilde{v})_i^n = \sum_p w_{ip}^n \cdot ((m_p \cdot \tilde{v})_p^n + \mathbf{Q}_p^n \cdot (\vec{x}_i - \vec{x}_p^n)) \quad (4.16 \text{ revisitada})$$

Per a realitzar això hem usat la matriu d'interpolació \mathbf{N} , definida a l'inici d'aquest apartat, per a incorporar iterativament el corresponent moment a les 3^3 cel·les a les quals pot afectar una partícula en 3D.

Ho podem visualitzar com un *kernel* 3D de mida $3 \times 3 \times 3$, amb el centre situat a la cel·la corresponent a la partícula.

Per obtenir el pes per interpolat cadascuna de les cel·les, hem de multiplicar les tres components de la matriu \mathbf{N} corresponents als índexs de les cel·les referents a la cel·la de la partícula, tal com l'equació 3.1 indica.

```
for (int i = -1; i < 2; ++i){
  for (int j = -1; j < 2; ++j){
    for (int k = -1; k < 2; ++k){
      // Nova cel·la
      Array3i cell_x = cell_i + Eigen::Array3i(i, j, k);
      // Vector partícula -> centre cel·la
      Vector3f cell_dist = d_size *
        ((cell_x.cast<float>() - (p.pos * grid_size)) + 0.5f);
      // Càlcul del pes
      float w = weights[i + 1].x() *
        weights[j + 1].y() *
        weights[k + 1].z();

      // Índex de la cel·la a la graella
      int index = getInd(cell_x.x(), cell_x.y(), cell_x.z());
      // 3 primeres components moment, 4a massa
      Array4f moment_mass = (Eigen::Array4f() <<
        p.v * mass,
        mass).finished();

      // Afegir Q al moment
      moment_mass.head<3>() += (affine * cell_dist).array();
      // Ponderar i sumar el moment
      grid[index] += w * moment_mass;
    }
  }
}
```

Cal remarcar que també s'interpola, en la quarta component de la cel·la, la massa de la partícula.

Important notar que `d_size` és la inversa de `grid_size`, per estalviar el cost de la divisió. També, la funció `getInd()` computa l'índex a la graella, donades les tres coordenades.

5.2.3. Processament de la graella

El processament de la graella és realment simple, perquè consisteix a iterar per cadascuna de les cel·les d'aquesta i aplicar senzilles transformacions. Codi complet a apèndix C.2.2.

Càlcul de la velocitat

En tota cel·la i , en què hi hagi massa no nul·la, hi haurà emmagatzemat un moment lineal $(m \cdot \tilde{v})_i^{n+1}$, de manera que, seguint l'explicat a la secció 4.1, podem calcular la nova velocitat \tilde{v}_i^{n+1} :

$$\tilde{v}_i^{n+1} = (m \cdot \tilde{v})_i^{n+1} / m_i^n \quad (5.6)$$

En aquest pas també podem afegir totes les forces externes, com la gravetat. Traduït a C++, aquesta operació es pot fer, donats els índexs i , j i k de la cel·la:

```
int idx = getInd(i, j, k);
Array4f& cell = grid[idx];
// cell.w() == mass
// Només si hi ha massa
if (cell.w() > 0){
    cell /= cell.w();
    cell.head<3>() += dt * g;
}
```

Límits de l'espai

Independentment de la mida de la graella, l'espai pel qual es poden moure les partícules és un cub de $1 \times 1 \times 1$, de manera que en cap cas una partícula hauria de sortir d'aquest.

En comptes de comprovar després de l'advecció d'una partícula si aquesta es troba fora de l'espai, és més senzill, i ràpid, modificar les velocitats de la graella per a assegurar que les partícules mai poden sortir d'aquests límits.

D'aquesta manera, durant la iteració per les cel·les de la graella, totes aquelles que estiguin a ϵ cel·les del límit de l'espai i tinguin una velocitat emmagatzemada amb direcció cap a l'exterior, tindran la seva respectiva velocitat anul·lada.

Per posar un exemple, donat l'eix vertical i un $\epsilon = 3$ podem comprovar, en funció de l'índex vertical j de la cel·la, la distància d'aquesta als límits, i decidir si eliminar la velocitat vertical.

```
if (j < 3 && cell.y() < 0.0f){
    cell.y() = 0.0f;
}
else if (j > grid_size - 3 && cell.y() > 0.0f){
    cell.y() = 0.0f;
}
```

Aquesta tècnica és útil per quan no es vol implementar un sistema de físiques de cap mena, o per separar aquest sistema de físiques del control dels límits de l'espai.

Col·lisions a graella

Per a tractar col·lisions amb objectes externs a la simulació, tal com es descriu a la secció 4.5, cal tenir alguna manera de discretitzar la superfície dels objectes a la graella.

La idea és que, de manera externa a aquest algorisme, la graella `physicsGrid` és omplerta amb la discretització de les normals de la superfície de l'objecte, per a tractar col·lisions amb objectes immòbils. D'aquesta manera, durant el processament de la graella, es pot comparar la direcció de la velocitat en la cel·la amb la superfície de l'objecte a aquesta. En cas que no hi hagi cap cos en aquesta cel·la, la normal serà 0.

Recordem que, abans de modificar la velocitat de la cel·la i , hem de comprovar que la normal i la velocitat relativa són oposades, és a dir: $\vec{v}_{i,\text{rel}} \bullet \hat{n}_{i,\text{obj}} < 0$, on $\vec{v}_{i,\text{rel}} = \vec{v}_i$ ja que l'objecte no té moviment. En cas afirmatiu, i de fregament nul, la nova velocitat $\vec{v}'_{i,\text{rel}}$ per objectes immòbils és:

$$\vec{v}'_{i,\text{rel}} = \vec{v}_{i,\text{rel}} - \hat{n}_{i,\text{obj}} \cdot (\vec{v}_{i,\text{rel}} \bullet \hat{n}_{i,\text{obj}}) \quad (5.7)$$

La qual cosa, en codi, pot ser implementada breument, a partir de l'índex `idx` de la cel·la en qüestió i la cel·la `cell`:

```
// Recuperar normal a la cel·la idx
Vector3f normalPhysics = physicsGrid[idx];
// dot(v_i, n_i)
float dot = normalPhysics.dot(cell.head<3>().matrix());
if (dot < 0.0f){
    // Editar velocitat, només tangencial
    cell.head<3>() = cell.head<3>() -
        (dot * normalPhysics).array();
}
```

Veure que si la normal és 0, quan no hi ha cap objecte, el producte escalar sempre donarà zero i mai es produiran col·lisions.

Per afegir col·lisions en moviment, s'hauria d'afegir una segona graella per emmagatzemar les velocitats de l'objecte discretitzades, ja que correspon a una altra magnitud vectorial, i dur a terme els càlculs amb la velocitat relativa entre les cel·les.

5.2.4. *G2P*

En aquesta última etapa es busca extreure de la graella la velocitat final de cada partícula, i calcular-ne la informació necessària per a computar les següents iteracions de la simulació.

La idea general és iterar per totes les partícules i per les cel·les veïnes a la seva posició, recuperant la velocitat calculada a l'anterior etapa de processament de la graella 5.2.3.

Cal dir que, com que les partícules no s'han mogut, els valors de la matriu d'interpolació **N** són idèntics als que s'han usat anteriorment, a la subsecció 5.2.2 de discretització de l'etapa *P2G*.

Nova velocitat

Per a calcular la nova velocitat, de la mateixa manera que es va emmagatzemar el moment, s'ha de recuperar usant els pesos de \mathbf{N} de la graella (secció 3.2.1) i usant la típica funció de *PIC* (secció 3.2.2).

$$\vec{v}_p^{n+1} = \sum_i w_{ip}^n \cdot \vec{v}_i^{n+1} \quad (3.5 \text{ revisitada})$$

Donada la partícula p , la seva posició a la graella `cell_i`, i la matriu de pesos `weights` (computats de la mateixa manera que a l'etapa *P2G* 5.2.2), calculem la nova velocitat:

```
p.v.setZero(); // Resetejar velocitat
for (int i = -1; i < 2; ++i){
  for (int j = -1; j < 2; ++j){
    for (int k = -1; k < 2; ++k){
      // cel·la veïna
      Array3i cell_x = cell_i + Array3i(i, j, k);

      float w = weights[i + 1].x() *
                weights[j + 1].y() *
                weights[k + 1].z();

      // Velocitat a la cel·la
      int idx = getInd(cell_x.x(), cell_x.y(), cell_x.z());
      Array3f cell_v = grid[idx].head<3>();
      // Interpolació
      p.v += w * cell_v;
    }
  }
}
```

Actualitzar *APIC*

Tal com s'ha definit a la secció d'*APIC* 4.2, el seu tensor \mathbf{C}_p ha de ser calculat en funció de la posició anterior de la partícula, més concretament la seva matriu \mathbf{B} de la descomposició $\mathbf{C}_p = \mathbf{B}_p \cdot (\mathbf{D}_p)^{-1}$, on \mathbf{D}_p^{-1} és una constant que ja sabem calcular.

Per altra banda, \mathbf{B}_p es defineix:

$$\mathbf{B}_p^{n+1} = \sum_i w_{ip}^n \cdot \vec{v}_i^{n+1} \cdot (\vec{x}_i - \vec{x}_p^n)^T \quad (4.4)$$

On l'operació $\vec{v}_i^{n+1} \cdot (\vec{x}_i - \vec{x}_p^n)^T$ dona lloc a una matriu quadrada, i és coneguda com el producte exterior de dos tensors.

És a dir:

$$\mathbf{B}_p^{n+1} = \sum_i w_{ip}^n \cdot \vec{v}_i^{n+1} \otimes (\vec{x}_i - \vec{x}_p^n) \quad (5.8)$$

Aquest tensor és calculable alhora que es du a terme la transferència de la nova velocitat, afegint la computació de la distància entre el centre de la cel·la i la partícula. En el següent extracte, els comentaris `/* -- */` al·ludeixen al codi mostrat anteriorment.

```
p.C.setZero(); // Resetejar C
for (/*i*/)for (/*j*/)for (/*k*/){
    // cel·la veïna
    Array3i cell_x = cell_i + Array3i(i, j, k);
    /* Càlcul velocitat, Eq. 4.19 */
    // Distància entre cel·la i partícula
    Vector3f cell_dist = cell_x.cast<float>() -
                        (p.pos * grid_size) + 0.5f;
    // Producte exterior, B * grid_size
    SumOuterProduct(p.C, w * cell_v, cell_dist);
}
// Multiplicar B per D^-1
p.C *= 4.0f * grid_size;
```

On `SumOuterProduct` és un mètode que donats dos vectors, en suma el producte exterior dels tensors al primer paràmetre (mètode a l'apèndix C.3).

Notar que com que hem dut a terme els càlculs en coordenades de graella, ja hem prèviament multiplicat les coordenades *Lagrangianes* per `grid_size`. Per tant només és necessari multiplicar altra vegada per `grid_size` i per 4 per obtenir $\mathbf{B}_p \cdot (\mathbf{D}_p)^{-1}$.

Advecció de la partícula

Un cop trobada la velocitat de la partícula en l'instant de temps actual, tot i que encara no hem actualitzat el gradient de deformació, ja podem provocar el moviment de la partícula en l'espai usant la velocitat calculada i el Δt de la simulació.

```
p.pos += dt * p.v;
```

On `dt` n'és la constant Δt .

Aquest pas és molt senzill, però és recomanable controlar molt aquesta nova posició per si de cas la simulació “explota”.

Actualització del gradient de deformació

Per actualitzar el gradient de deformació s'usa l'aproximació que *MLS* ens aporta (secció 4.3.1) usant el tensor \mathbf{C} d'*APIC* calculat en aquesta mateixa iteració:

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \cdot \mathbf{C}_p^{n+1}) \cdot \mathbf{F}_p^n \quad (4.8 \text{ revisitada})$$

Actualització que es pot implementar senzillament en C++:

```
Matrix3f F = (Matrix3f::Identity() + (dt * p.C)) * p.F;
```

De manera que aquest tensor \mathbf{F} indica la nova deformació, tot i que inclou també una part de deformació plàstica, la qual ha de ser tractada.

Aplicació de la plasticitat

Tal com s'ha vist a la secció 4.4, donat un gradient de deformació \mathbf{F} que inclou deformació plàstica, i un determinant J_P corresponents al gradient de deformació plàstica de l'instant de temps anterior, podem computar un nou gradient amb només la part elàstica, i actualitzar aquest determinant.

Per això descomponem el gradient, n'extraïem l'elongació plàstica, i el tornem a construir.

```
// SVD del gradient F
JacobiSVD<Matrix3f, NoQRPreconditioner>
    svd(F, ComputeFullU | ComputeFullV);
Vector3f svd_e = svd.singularValues();
// Clamp segons t_c i t_s
for (int i = 0; i < 3; ++i) {
    svd_e[i] = clamp(svd_e[i], 1.0f - t_c, 1.0f + t_s);
}
// F elàstic
p.F = svd.matrixU() *
    svd_e.asDiagonal() *
    svd.matrixV().transpose();
```

On les variables t_c i t_s corresponent a les constants de plasticitat θ_c i θ_s respectivament. Concretament, els valors $2.5 \cdot 10^{-2}$ i $7.5 \cdot 10^{-3}$ s'han vist a [5] produir bons resultats per a simular neu.

Per actualitzar el determinant J_P , usem l'equació següent:

$$J_P^{n+1} = (\tilde{J}_E^{n+1} \cdot J_P^n) / J^{n+1} \quad (4.25 \text{ revisitada})$$

On \tilde{J}_E^{n+1} és el determinant de \mathbf{F} abans d'aplicar-li la plasticitat. És a dir:

```
const float oldJ = F.determinant();
/* Càlcul de F elàstic, Eq. 4.8 */
float newJp = oldJ * p.Jp / p.F.determinant(); // Eq. 4.25
p.Jp = clamp(newJp, p_c, p_s);
```

On l'operació `clamp` final limita la deformació plàstica entre dues constants p_c i p_s . Uns bons valors per aquestes són 0.6 i 20, respectivament.

5.3. Optimitzacions

Donat l'algorisme principal ja explicat en l'anterior secció, hi ha diverses optimitzacions que es poden fer, i altres factors a tenir en compte separats de l'algorisme principal.

5.3.1. Paral·lelisme

Paral·lelitzar el codi suposa una gran optimització en màquines multi-core, per això és un dels primers punts de millora del sistema.

L'etapa de $G2P$ és senzillament paral·lelitzable, ja que no hi ha cap mena de *race-condition*. Això és degut al fet que només es duen a terme lectures de la graella, i l'actualització de la mateixa partícula de la iteració actual. D'aquesta manera, podem usar la llibreria d'*OpenMP* [23] per a paral·lelitzar el `for` que marca la iteració sobre les partícules de l'etapa $G2P$.

Evidentment, l'etapa de processament de la graella també és paral·lelitzable de la mateixa manera, tot i que no s'han observat millores considerables en la paral·lelització d'aquesta.

Finalment, per paral·lelitzar l'etapa de $P2G$ ja es necessita tenir en compte *race-conditions*, doncs pot ser que dues partícules actualitzin la mateixa cel·la de la graella alhora.

Tot i que podríem usar un sistema de *locks* per a cada cel·la, això no és eficient. D'aquesta manera, Hu et al. a [24] proposen un sistema de divisió de la graella en diversos blocs, i paral·lelitzar totes les transferències en aquests blocs sempre que siguin independents.

Aquesta característica no es troba implementada al codi desenvolupat.

5.3.2. Optimització $P2G$

Una optimització algorítmica que es pot efectuar en el bucle de transferència del $P2G$, és veient que la multiplicació de la matriu afí \mathbf{Q} pel vector distància partícula-cel·la varia a cada iteració en un valor constant, ja que aquesta distància canvia en 1 unitat a cada iteració en la mateixa dimensió, ja que les cel·les són de mida 1.

De manera que recordant l'equació 4.16, podem definir la transferència de moment lineal d'una partícula p a una cel·la i :

$$(m \cdot \tilde{v})_{i,p}^n = w_{ip}^n \cdot ((m_p \cdot \tilde{v})_p^n + \mathbf{Q}_p^n \cdot (\vec{x}_i - \vec{x}_p^n)) \quad (5.9)$$

D'aquesta forma, veiem que l'únic terme que varia entre iteracions és el vector que multiplica \mathbf{Q} , concretament $(\vec{x}_i - \vec{x}_p^n)$.

Així podem descompondre el tensor, aïllant aquesta variable el màxim.

$$\begin{aligned}\mathbf{Q}_p^n \cdot (\vec{x}_i - \vec{x}_p^n) &= \mathbf{Q}_p^n \cdot (\tilde{x}_p^n + \vec{\epsilon} - \vec{x}_p^n) \\ &= \mathbf{Q}_p^n \cdot (\tilde{x}_p^n - \vec{x}_p^n) + \mathbf{Q}_p^n \cdot \vec{\epsilon}\end{aligned}\quad (5.10)$$

On \tilde{x}_p^n és el centre de la cel·la on p pertany, i el vector $\vec{\epsilon}$ manté la invariant $\vec{x}_i = \tilde{x}_p^n + \vec{\epsilon}$, de manera que és la diferència entre la cel·la de la partícula i la cel·la on discretitzar la informació.

Aquesta descomposició és important, ja que sabem que les components del vector $\vec{\epsilon}$ sempre seran $\{-1, 0, 1\}$, ja que només es durà a terme transferència entre les cel·les veïnes a \tilde{x}_p^n .

De la mateixa manera podem descompondre $\mathbf{Q}_p^n \cdot \vec{\epsilon}$ segons les seves 3 components $i, j, k \in \{-1, 0, 1\}$:

$$\mathbf{Q}_p^n \cdot \vec{\epsilon} = i \cdot \mathbf{Q}_p^n \cdot \hat{i} + j \cdot \mathbf{Q}_p^n \cdot \hat{j} + k \cdot \mathbf{Q}_p^n \cdot \hat{k} \quad (5.11)$$

On $\hat{i}, \hat{j}, \hat{k}$ són els vectors unitaris de la base. Com que \mathbf{Q}_p^n i els vectors unitaris estan en la mateixa base, és equivalent usar la corresponent columna del tensor $\mathbf{Q}_p^n = \begin{pmatrix} \vec{Q}_{p,\hat{i}}^n & \vec{Q}_{p,\hat{j}}^n & \vec{Q}_{p,\hat{k}}^n \end{pmatrix}$ directament.

$$\mathbf{Q}_p^n \cdot \vec{\epsilon} = i \cdot \vec{Q}_{p,\hat{i}}^n + j \cdot \vec{Q}_{p,\hat{j}}^n + k \cdot \vec{Q}_{p,\hat{k}}^n \quad (5.12)$$

Usant aquesta descomposició, podem canviar les multiplicacions de matriu-vector per una de sola $\mathbf{Q}_p^n \cdot (\tilde{x}_p^n - \vec{x}_p^n)$ i afegir la resta de components mitjançant sumes i multiplicacions d'escalars-vectors.

El codi en C++ consisteix en:

```
// Primera cel·la (centre) de les iteracions
Array3i cell_x0 = cell_i + Array3i::Constant(-1);
// Vector partícula-Centre cel·la
Vector3f cell_dist0 =
    ((cell_x0.cast<float>()) - (p.pos * grid_size)) + 0.5f);

// Moment inicial, sense Q
Array4f moment_mass0 = (Array4f() <<
    p.v * mass, mass).finished();
// Multiplicar per d_size per passar a [0,1]
// Q * dist
moment_mass0.head<3>() += d_size * (affine * cell_dist0).array();
Array3f kstep = affine.col(2) * d_size;
Array3f jSemiStep = (affine.col(1) * d_size).array();
// Reiniciar a k = 0
Array3f jstep = jSemiStep - (3.0f * kstep);
// Reiniciar a j = k = 0
Array3f iSemiStep = (affine.col(0) * d_size).array();
Array3f istep = iSemiStep - (3.0f * jSemiStep);
```

On s'ha usat com a base la primera cel·la de la iteració, de manera que les components de $\vec{\epsilon}$ són en $\{0, 1, 2\}$.

És important veure que les variables `jstep` i `istep` eliminen les addicions de les iteracions més internes, de manera que garanteixen dur a terme totes les permutacions.

El bucle intern de la transferència passa a ser el següent:

```
for (int i = -1; i < 2; ++i){
  for (int j = -1; j < 2; ++j){
    for (int k = -1; k < 2; ++k){

      const float w = weights[i + 1].x() *
                      weights[j + 1].y() *
                      weights[k + 1].z();
      const int index = getInd(cell_i.x() + i,
                              cell_i.y() + j,
                              cell_i.z() + k);

      grid[index] += w * moment_mass0;

      moment_mass0.head<3>() += kstep;
    }
    moment_mass0.head<3>() += jstep;
  }
  moment_mass0.head<3>() += istep;
}
```

Aquesta és una optimització important, ja que donat que cada multiplicació matriu-vector costa 15 *FLOPs* (*float operations*), i aquesta es duia a terme 3^3 vegades, de manera que s'executaven 405 *FLOPs* per calcular \mathbf{Q} , i 3^3 addicions de vectors (3 *FLOPs* per addició) per la contribució al moment lineal; un total de 486 *FLOPs*.

Amb aquest canvi, duem a terme només una multiplicació matriu-vector, 5 multiplicacions escalar-vector (3 *FLOPs* per producte) i $3^3 + 3$ addicions o subtraccions, les quals ja computen la contribució de \mathbf{Q} al moment lineal; de manera que obtenim un cost total de 270 *FLOPs*, una reducció del 44.4% d'operacions.

En els tests realitzats, aquesta optimització s'ha vist reflectida en un x1.16 de *speedup* sobre el temps total.

De manera semblant es pot optimitzar l'etapa de *G2P*, computant les diferències entre iteracions, en comptes de la nova posició cada vegada. Consultar apèndix [C.2.3](#).

5.3.3. Desenroscat de bucles

La tècnica del desenroscat de bucles (*loop unrolling*) consisteix a explicitar totes les iteracions d'un bucle.

En el nostre cas, és una bona optimització a aplicar en els dos bucles del codi de transferència de dades entre partícules i graella, doncs explicita totes les operacions estalviant-se tots els salts en els bucles.

Usualment els compiladors no desenrosquen bucles tan llargs (27 iteracions), però amb l'opció de compilació per *GCC #pragma GCC unroll* obliguem a que aquesta optimització es dugui a terme de manera automàtica.

Aquest detall ens estalvia la creació de petites estructures de dades, que radica en un *speedup* d'un 3% aproximadament, en relació amb la versió ja optimitzada.

5.3.4. Graella paginada esparsa

Hu et al. al document [24] proposen usar una graella paginada esparsa, o *SP-Grid*, de Setaluri et al. [25] com a estructura de dades per la graella.

Aquesta empra l'ordre de Morton i les característiques del *TLB* del processador per maximitzar la localitat de les dades, i minimitzar la latència dels accessos.

Aquesta característica no ha estat implementada, perquè es basa en una anàlisi a baix nivell del hardware, és dependent de l'execució especulativa del processador, i per tant no acaba d'entrar en l'àmbit d'aquest treball; tot i que és una bona optimització a aplicar.

5.4. Materials

No tots els sistemes a simular seran homogenis pel que fa al material, sinó que cadascuna de les partícules tindrà propietats diferents corresponents al seu material original.

Entre aquestes propietats no només hi ha la massa i el volum, sinó també els paràmetres de Lamé i la l'enduriment, per exemple.

Tot i que és possible, no resulta eficient copiar totes aquestes dades a les pròpies partícules, doncs estarem replicant unes poques dades moltes vegades, fet que suposarà un *overhead* de memòria, reflectit sobretot en *cache*.

La solució és crear una estructura de materials externa a l'algoritme, i que cada partícula mantingui un punter al seu respectiu material, de manera que cada un és autocontingut.

En concret emmagatzema tot el següent:

- Modul de *Young*, o *E*: Constant de força necessària per aplicar la deformació.
- Coeficient de *Poasson*, o ν : Relació entre l'elongació tangencial i axial.

- μ , o μ : Paràmetre de Lamé computat a partir de E i ν .
- λ , o λ : Paràmetre de Lamé computat a partir de E i ν .
- Volum: El volum que ocupa una partícula.
- Massa, o m : La massa de la partícula.
- Enduriment (*hardening*), o \mathcal{E} : Coeficient de duresa de la partícula per la plasticitat.
- t_c , o θ_c : Defineix el punt de màxima deformació elàstica per compressió.
- t_s , o θ_s : Defineix el punt de màxima deformació elàstica per elongació.
- p_c : Defineix el mínim canvi de volum de la deformació plàstica.
- p_s : Defineix el màxim canvi de volum de la deformació plàstica.

D'aquesta manera, les simulacions són molt més configurables i versàtils.

5.5. Complexitat

Una de les parts més positives de l'algoritme de *MPM* és que és lineal respecte al nombre de partícules a simular.

Aquest fet és senzill de veure en els passos de *P2G* i *G2P*, on només s'itera una vegada per cada partícula, transmetent dades entre ella i la graella. També cal dir que la mida de la graella també influeix linealment en la complexitat de l'algoritme, ja que durant l'etapa de processament de la graella s'han de recórrer una vegada totes les cel·les.

D'aquesta manera, si n és el nombre de partícules i m el nombre de cel·les de la graella, la complexitat final és de $\Theta(n + m)$ per cada pas de *MPM*.

De totes formes, els principals colls d'ampolla de l'algorisme són tant el *P2G* com el *G2P*, a causa de la quantitat de *FLOPs* a efectuar, i en el cas d'usar el model corrotacional fix cal tenir en compte el cost de les descomposicions en valors singulars.

En aquest últim cas del model corrotacional fix, la complexitat de l'algorisme va molt lligada a la simulació que s'ha de realitzar, doncs com més es diferenciïn els gradients de deformació de la identitat, més costosa serà la descomposició en valors singulars.

6. Interfícies

Donat el simulador creat i explicat en els anteriors capítols, un usuari ha de poder interactuar amb aquest *software*, i ser capaç de crear, visualitzar, exportar i emmagatzemar diferents simulacions.

Totes aquestes interfícies d'usuari es troben explicades i desenvolupades en aquest capítol, així com les seves dependències, motivació i especificació.

6.1. Entrada de dades

L'entrada de dades es realitza per l'entrada estàndard, en pro de poder usar scripts i planificar l'execució de simulacions còmodament.

Aquesta entrada configura tot el sistema a simular, permetent primer de tot escollir el model energètic a usar durant la simulació d'entre corrotacional fix (secció 2.5.3), *Neo-Hookean* (secció 2.5.2), i un model anomenat *Sand* que només usa *APIC* (secció 4.2). D'aquesta manera el simulador s'adaptarà completament a aquest model energètic.

Per altra banda, s'accepten un nombre indeterminat de materials diferents per a configurar tots aquells cossos diferents a simular. Cada material ve definit per les característiques esmentades a la secció anterior 5.4.

Posteriorment a introduir el nombre de partícules en la simulació, s'ha creat una llista de models a carregar, que s'ompliran de partícules, i sobre els que es durà a terme la simulació. Cadascun d'aquests models permet uns paràmetres diferents, com velocitat, i diversos materials.

L'activació de físiques és opcional, ja que no en totes les simulacions es vol experimentar amb aquestes. D'activar-les, es pot escollir entre diversos cossos a col·lidir durant la simulació.

Finalment, donat un nom d'arxiu, s'emmagatzemarà la simulació en un arxiu *sbf* (secció 6.2), i es demanarà el nombre de fotogrames a efectuar de la simulació juntament amb el temps entre aquests en fotogrames per segon.

6.1.1. Models de simulació

Donat l'escàs temps que dura l'elaboració d'aquest treball, no ha estat possible implementar la càrrega de models en malla de triangles a la simulació, tot i que s'explica a continuació.

Per a omplir una malla de triangles amb partícules, després de carregar la *mesh* al sistema, cal voxelitzar aquesta usant, per exemple, un *Octree*. D'aquesta manera delimitem eficientment les fronteres de l'objecte, i obtenim vòxels fàcilment tractables a omplir amb partícules homogèniament.

En el cas d'aquest projecte, donat un nombre de partícules variable, s'han programat cossos a aproximar que permeten visualitzar les propietats d'aquest simulador amb comoditat; el codi d'aquesta característica es troba en la capçalera `ParticleStructures.h`.

En concret els objectes són els següents:

- **Box Filled:** Cub allargat omplert de manera aleatòria, ocupant un ampli espai de la zona de simulació.
- **Box Filled Homogen:** El mateix cub anterior, però omplert de manera homogènia.
- **3 boxes different heights:** Tres cubs situats en diferents altituds, amplituds i profunditats de l'espai.
- **3 boxes aligned:** Tres cubs situats en la mateixa altitud i profunditat.
- **Sphere coliding:** Dues esferes que col·lideixen entre elles.
- **C:** Una construcció amb forma de lletra 'C', de manera que la part superior pot ser tractada com un material afectat per la gravetat.

6.1.2. Models de físiques

De la mateixa manera que a l'apartat anterior, s'han implementat uns objectes immòbils a ser utilitzats pel sistema a simular, també es poden afegir objectes a interaccionar de manera externa amb la simulació.

Els models creats són senzills, simplement per observar com funciona la interacció amb aquest tipus de col·lisions. Aquests són els següents:

- **Flat:** Cos de prova, en què es crea una superfície plana amb alçada configurable.
- **Two slopes:** Es creen dues rampes en diagonal, com un rellotge de sorra.
- **Two slopes unbounded:** Idèntic a l'anterior, però sense límits en la profunditat.

6.2. Emmagatzematge

És important emmagatzemar les simulacions alhora que es duen a terme, en algun format que després en permeti la seva correcta visualització.

El format imatge o vídeo no és acceptable, ja que es perd molt de detall de la simulació que no es pot apreciar. D'aquesta manera s'ha optat per crear un format binari propi.

Un arxiu *sbfi*, acrònim de *simulation binary file*, és un arxiu binari que emmagatzema les posicions de totes les partícules durant els diferents fotogrames de la simulació.

També emmagatzema totes les dades de la simulació, així com els diferents materials i el color de les partícules.

Aquest fitxer s'ha estructurat en etiquetes sense ordre, de manera que precedint a qualsevol mena de dada sempre hi haurà una etiqueta que la defineix, i de la qual se'n pot extreure el tipus i l'extensió de la dada. Per a poder extrapolar bé la longitud d'aquestes, és necessari que sempre la primera dada de l'arxiu sigui el nombre de partícules en la simulació.

L'emmagatzematge de fitxers es troba en la classe de `WriteSBF`, mentre que la lectura d'aquests fitxers es troba en la classe de `ReadSBF`.

Aquests arxius han sigut creats per a ser intercanviables entre sistemes Debian i Windows, per evitar problemes de compatibilitat, i per a ser comprimits si es requereix estalviar espai de disc.

Aquest sistema també permet que sigui possible, de ser necessari, crear noves etiquetes i emmagatzemar més dades de la simulació, tant constants d'aquestes, com propietats de les partícules individualment, per exemple les velocitats o els gradients de deformació, de manera que es podria dur a terme simulacions discontinuades.

La lectura d'aquests és basa en imprimir primerament els paràmetres de la simulació pel terminal, i posteriorment guardar tots els fotogrames en una estructura de dades interna, la qual serà usada pel visualitzador.

6.3. Visualitzador

Per a poder comprovar els resultats de les simulacions, aquestes s'han de poder visualitzar de manera entenedora a l'ull humà, doncs és l'única manera de verificar que una simulació és gràficament realista.

Així s'ha creat un visualitzador en 3 dimensions, amb càmera mòbil i comandes diverses, per poder observar la simulació amb tot detall.

S'ha usat la llibreria d'*OpenGL*[22], la qual aporta una interfície de programació per a renderitzar gràfics tridimensionals, és multiplataforma, i s'adapta a les necessitats d'aquest projecte.

D'altra banda, s'ha usat la llibreria *Open Source GLFW* [26] per a la creació de finestres, independentment del sistema operatiu usat, i per tractar esdeveniments i l'entrada de l'usuari (teclat i ratolí).

I per integrar *OpenGL* amb els controladors del sistema i la targeta gràfica, s'ha usat *GLAD*[27].

Aquest visualitzador està format per diferents components, implementades a la classe **SimVisualizer**, de manera que distingim la renderització de partícules, el dibuixat de fotogrames i la interacció d'usuari.

6.3.1. Renderització

A l'hora de renderitzar les partícules, s'ha de tenir en compte no només l'eficiència d'aquest procediment, sinó que sigui visualment entenedor del que succeeix en la simulació, de manera que s'han de poder observar els conceptes de volum, profunditat i moviment.

Per transformar les partícules, que es defineixen únicament per una posició, a algun objecte tridimensional s'han usat icosaedres regulars, un poliedre de 20 cares, ja que és una bona i simple aproximació a una esfera, on els vèrtexs, les cares i les respectives normals són senzilles de computar (funció **genIcosphere()**, consultable a l'apèndix C.1).

Com que hi ha un nombre molt gran de partícules a renderitzar, i es vol evitar dur a terme un dibuixat per cadascuna d'elles, es du a terme un dibuixat instanciat, anomenat en *OpenGL* **glDrawArraysInstanced**. Aquesta crida, donat un model, emet una única crida a la gràfica que dibuixa, en el nostre cas, tants icosaedres com partícules però en posicions diferents identificades per un altre *array*.

Amb l'objectiu de donar encara més la il·lusió de què els icosaedres són esferes, s'afegeix una llum a l'espai de manera que les partícules s'il·luminen d'acord amb aquesta, usant en concret un model local d'il·luminació amb reflexió de llum difusa Lambertiana.

$$\vec{c}_{\text{shaded}} = \vec{c}_{\text{obj}} \cdot (\vec{c}_{\text{amb}} + (\hat{l} \bullet \hat{n})^+ \cdot \vec{c}_{\text{dif}}) \quad (6.1)$$

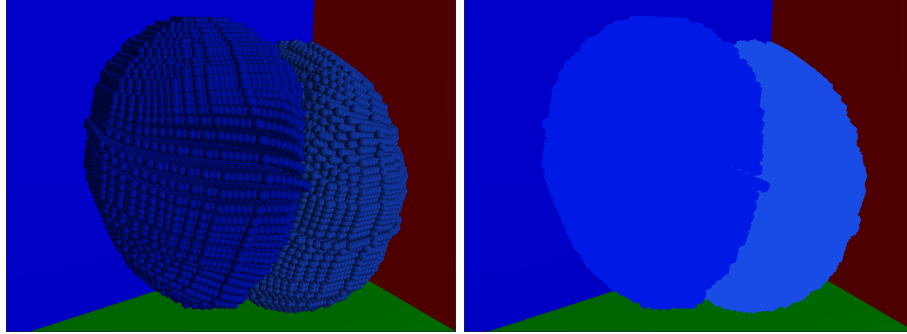


Figura 6.1: *Comparació ombrejat local*. Mateix fotograma d'una simulació, amb (imatge esquerra) i sense (imatge dreta) aplicar ombreig local. Simulació B.4.

On \vec{c}_{shaded} és el color final, \vec{c}_{obj} el color de la superfície de l'objecte a renderitzar (partícula en aquest cas), \vec{c}_{amb} i \vec{c}_{dif} el color de la llum ambient i difusa, respectivament. Els vectors normalitzats \hat{n} i \hat{l} corresponen a la normal en el punt actual, i a la direcció de la llum respecte d'aquest punt. Finalment, la notació $(\hat{l} \bullet \hat{n})^+$ indica l'operació *clamp* sobre nombres negatius, de manera que el resultat mínim és zero.

A més a més, s'assigna com a normals dels vèrtexs de l'icosaedre la mitjana de la normal de les cares adjacents, de manera que durant la renderització es du a terme una interpolació d'aquestes, aconseguint una similitud important a una esfera.

Per a donar una idea d'espai es defineix l'entorn com un cub, de manera que tota simulació es durà a terme o es reproduirà a dins d'aquest. L'objectiu és que només les cares oposades a la càmera siguin visibles, de manera que facin de superfície de la simulació.

Aquesta característica es pot implementar activant el *front face culling* durant el dibuixat del cub, de manera que només les cares amb normals oposades al vector direcció es veuran per pantalla.

Com que tractem amb superfícies planes, és adient aplicar un model de reflexos especulars per donar una millor idea de la posició de la llum en l'entorn. Aquest es basa en calcular el vector reflex \hat{r} de la llum sobre la superfície a ombrejar,

$$\vec{c}_{\text{shaded}} = \vec{c}_{\text{obj}} \cdot (\vec{c}_{\text{amb}} + (\hat{l} \bullet \hat{n})^+ \cdot \vec{c}_{\text{dif}} + (\hat{r} \bullet \hat{v})^{+\alpha} \cdot \vec{c}_{\text{spec}}) \quad (6.2)$$

On \hat{v} és la direcció de la càmera, c_{spec} la llum especular, i α un paràmetre que configura com d'especular és el material, en el cas del cub és 80.

Cal recordar que \hat{r} és calculable de la següent manera:

$$\hat{r} = 2 \cdot (\hat{l} \bullet \hat{n}) \cdot \hat{n} - \hat{l} \quad (6.3)$$

Com es pot observar a la figura 6.1, és imprescindible aplicar tècniques d'ombreig per a poder percebre qualsevol canvi de forma o densitat en el cos.

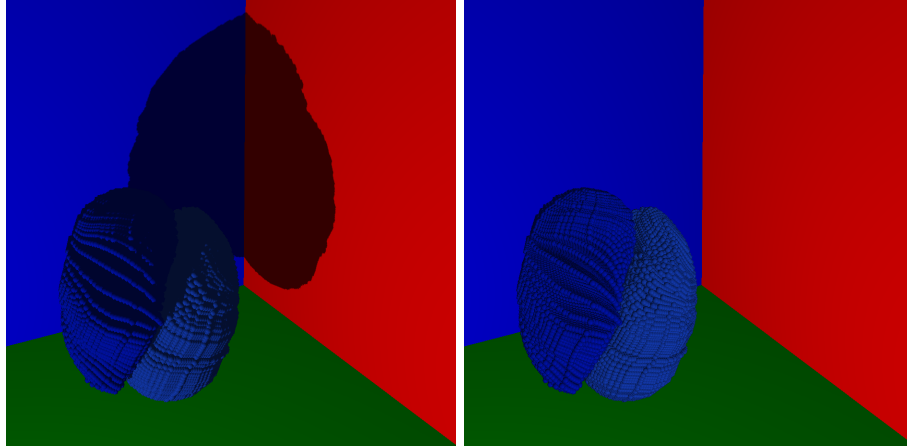


Figura 6.2: *Comparació shadow map*. Mateix fotograma d'una simulació, amb (imatge esquerra) i sense (imatge dreta) aplicar el *shadow map*, notar la diferencia d'apreciació de la profunditat i volum. Simulació [B.4](#).

No obstant aquest ombreig, és difícil tenir el concepte de profunditat o volum, doncs no existeixen oclusions visuals entre objectes.

Per això s'ha usat la tècnica de *shadow mapping*, donant a les partícules la capacitat de causar ombres a altres partícules, i al cub englobant.

Aquesta tècnica consisteix a renderitzar l'escena des del punt de vista de la llum a cada fotograma, i emmagatzemar la profunditat de l'escena en una textura, de manera que aquesta té la informació del punt sòlid més proper a la llum a cada fotograma. Llavors, a l'hora d'ombrejar un fragment d'un objecte, es comprova si aquest fragment des del punt de vista de la llum es troba a més profunditat que el corresponent *texel* emmagatzemat a la textura.

Els efectes de l'aplicació d'aquest *shadow map* es poden observar a la figura [6.2](#), on aquest afegeix més informació de volum, posició de la llum, i intuïció de la distància entre cossos a la imatge.

6.3.2. Fotogrames

Un cop carregada una simulació d'un arxiu *sbf* (secció [6.2](#)), aquesta ha de ser mostrada a l'usuari.

Els fotogrames són carregats en una estructura de dades interna, que manté un seguiment de les posicions de les partícules, i a part s'emmagatzemen les propietats d'aquestes partícules, com el color assignat.

Donats els fotogrames, s'envien a la gràfica cíclicament de manera que la simulació es reinicia al final d'aquesta. Entre fotogrames, es segueixen duent a terme renderitzacions, per acceptar entrada de l'usuari entre aquests (com es veurà a la següent secció 6.3.3) i actualitzant la posició de la càmera.

És a dir, s'ha programat un sistema d'esdeveniments els quals activen el canvi de fotograma, amb un temps configurable entre aquests, de manera que no bloquegen les interaccions amb l'usuari.

6.3.3. Interacció

La interacció amb l'usuari es troba diversificada, perquè inclou tan entrada per teclat o ratolí, i no tota acció corresponent a una entrada es troba controlada pel visualitzador, però si activada per aquest. És a dir, el visualitzador controla tota l'entrada al sistema, però en delega les aplicacions mitjançant *callbacks*.

Aquesta decisió de disseny és perquè únicament aquesta classe depengui de *GLAD*, que és la llibreria que controla l'entrada al sistema, evitant possibles problemes d'incompatibilitats que poguessin sorgir, i duplicat de codi.

Concretament, el visualitzador reserva el ratolí per a moure la càmera en l'entorn de la simulació, juntament amb les tecles **A**, **S**, **D**, **W** pel moviment axial, la tecla *mayúscula* per reduir la velocitat de moviment de la càmera, i la tecla **ESC** per a tancar correctament el visualitzador, i la simulació si és que s'estava executant.

És important mencionar que s'usen les tecles **M** i **N** per a focalitzar i des-focalitzar la finestra, de manera que es desactiven totes les interaccions amb aquesta, permetent usar el ratolí fora de l'entorn, escalar la finestra, o introduir dades per consola.

Les tecles **R** i **T** recarreguen les *shaders*, i l'última activa o desactiva el *shadow map*. Finalment, els símbols **+** i **-** del teclat numèric modifiquen la mida de les partícules, i la tecla **F** canvia la posició de la llum a posició de la càmera.

Per altra banda, totes aquelles tecles no reservades poden ser configurades per a cridar qualsevol mètode que es desitgi en forma de funcions lambda.

Per exemple, les tecles **K** i **L** incrementen o redueixen el temps entre fotogrames, la tecla **P** pausa la reproducció de la simulació i **Enter** la reprèn, i **O** reinicia la reproducció amb el primer fotograma. Finalment la tecla **I** inicia l'exportació de la simulació.

6.4. Exportació

Per exportar les simulacions s'ha escollit el format *gif*, el qual permet fer un petit vídeo repetitiu a partir d'imatges, la qual cosa és adient per aquest objectiu.

L'exportació s'ha realitzat usat la llibreria *Open Source* de Tangora *gif-h*[\[28\]](#), la qual permet generar *gif* a partir de *buffers* de color RGBA.

D'aquesta manera, donat un posicionament de la càmera, i la resolució de la finestra, es genera aquest petit vídeo.

És possible que aquest *gif* hagi de ser postprocessat per aconseguir la velocitat adient, resolució pertinent, o de ser necessari haver de comprimir el mateix arxiu.

7. Resultats

Donada tota l'explicació dels capítols anteriors, les implementacions del simulador i de totes les interfícies, ens trobem amb un *software* que pot ser considerat complet.

En aquest capítol es mostren les capacitats d'aquest programa, i la qualitat de les seves simulacions, juntament amb diversos exemples de les diferents característiques que poden ser simulades de manera visual, doncs és l'única forma de poder validar els resultats correctament.

Totes les simulacions referenciades en aquest apartat, i moltes altres que poden resultar interessants, estan disponibles en l'apèndix B juntament amb els paràmetres de la mateixa, per facilitar la replicació de les mateixes simulacions de ser necessari.

Informativament, totes les simulacions han estat dutes a terme amb una graella de $128 \times 128 \times 128$.

7.1. Temps

El temps que triguen les simulacions es veu directament influït per la constant Δt , la qual marca els intervals entre passos de la simulació, tal com s'ha vist en el capítol 5, de manera que Δt més grans permetran obtenir simulacions més ràpidament, ja que seran requerides menys iteracions de l'algorisme per cada fotograma.

Però no podem assignar a aquest paràmetre un valor arbitràriament gran, doncs aquest també influirà en la precisió dels mètodes numèrics emprats.

En aquest treball, s'ha estat implementat l'algorisme de *MPM-MLS* amb el mètode d'integració explícita, de manera que les simulacions tendeixen a expandir-se i, en determinats casos, a “explotar”.

Per exemple, en la simulació de la figura 7.1 es pot veure un d'aquests casos en que Δt és massa gran.

D'aquesta manera s'ha escollit finalment un $\Delta t = 10^{-5}s$, suficientment petit per evitar aquests errors, però prou gran per a que la simulació sigui tractable. En simulacions a 60 fotogrames per segon, aquest Δt fa que cada fotograma equivalgui a gairebé 1675 iteracions de l'algoritme, o a gairebé 1350 si s'usen 30 fotogrames per segon.

De qualsevol manera, cada segon de la simulació equival a 10^5 iteracions.

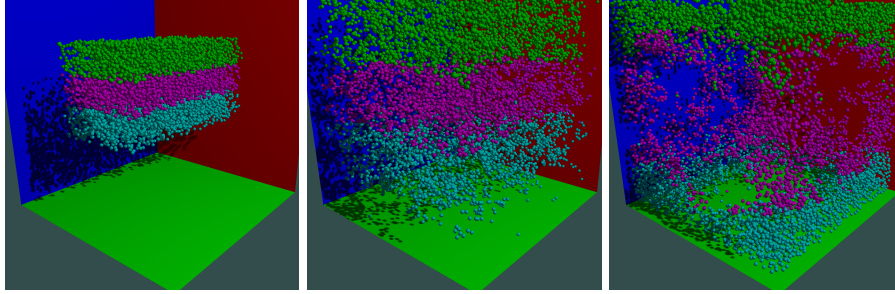


Figura 7.1: *Explosió d'una simulació*. Fotogrames progressius de la simulació *explode* (apèndix B.2), d'esquerra a dreta, amb un $\Delta t = 3 \cdot 10^{-5} s$, no suficientment petit per evitar que l'integració explícita provoqui aquests errors.

Simulació	Fotograma	Corrotacional	Neo-Hookean
<i>C7, C8</i> (105316)	30	77 s	51 s
	52	80 s	52 s
	80	60 s	52 s
<i>nPlastic6, nPlastic7</i> (80776)	15	108 s	48 s
	50	112 s	48 s
	80	105 s	46 s
<i>hard5, hard6</i> (80776)	20	74 s	27 s
	50	82 s	28 s
	80	83 s	28 s

Taula 7.1: *Temps de simulació*. Temps de simulacions de cada fotograma en diferents simulacions, diferenciant models corrotacional fix i *Neo-Hookean*. Les simulacions especificades és poden trobar a l'apèndix B. El nombre entre parèntesis identifica el nombre de partícules. Elaboració pròpia.

Per altra banda, per justificar el temps de simulació, cal veure que aquest és molt variable, doncs en un instant inicial les transferències partícula-graella seran en zones properes, i la descomposició en valors singulars serà ràpida, a causa de la semblança del gradient de deformació amb la matriu identitat.

Segons progressa la simulació, si el fluid o cos es trenca o es dispersa, incrementarà la quantitat de transferències partícula-graella diferents, així com la complexitat de la *svd*.

Aquests fets provoquen que, per exemple, una simulació de 10^4 partícules trigui 26 segons per cada fotograma, mentre que després d'unes certes iteracions trigui una mitjana de 60 segons per cada fotograma.

De manera que el factor temps no és només dependent del nombre de partícules, sinó de la mateixa simulació.

Tot i això, per donar uns valors temporals orientatius, s'han calculat els temps de còmput per fotograma de diferents simulacions, plasmats a la taula 7.1. Els temps són comparant fotogrames amb alta deformació, i diferents models.

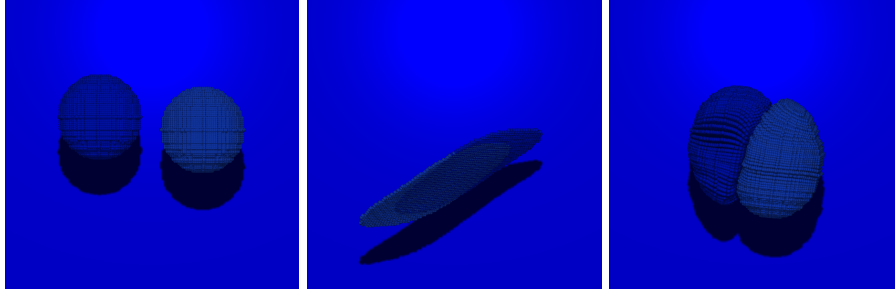


Figura 7.2: *Acció del model elàstic.* Primera imatge, simulació amb dues esferes a punt de col·lidir. Imatge central, mateixa simulació de la primera imatge uns fotogrames després, amb un material sense cap mena de model d'hiperelasticitat; imatge dreta, mateixa simulació però amb model corrotacional fix. Simulacions *noQ* i *noQY*, apèndix B.3 i B.4.

Es pot observar com el model *Neo-Hookean* és temporalment molt estable, sobretot en comparació amb el corrotacional fix. Això és degut a les computacions de la descomposició en valors singulars, i a una quantitat de productes de matrius a computar superior als de l'altre model.

També és important veure com en diversos models, tot i tenir la mateixa quantitat de partícules, els temps són molt variants pel motiu esmentat anteriorment.

7.2. Resultats de les simulacions

En aquesta secció es desenvolupen totes les particularitats i fenòmens que es poden simular amb el simulador implementat en aquest treball.

7.2.1. Models potencials elàstics

En aquest treball s'han presentat els models *Neo-Hookean* i corrotacional fix, seccions 2.5.2 i 2.5.3 respectivament, de manera que s'han dut a terme diferents simulacions entre els dos.

Primerament cal veure que aquest model potencial elàstic és el que acaba mantenint la forma del cos, de manera que és absolutament necessari el fet de triar alguna funció energètica.

Observant la figura 7.2 veiem que sense model, un cos no és capaç de mantenir la seva forma de cap manera.

Per altra banda, caldria diferenciar aquests dos models elàstics explicats en aquest document.

Tot i les diferències en temps de computació, els resultats són, en la gran majoria dels casos provats, molt similars, només produint diferències notables en deformacions molt grans, en les que el model corrotacional és capaç de conservar millor l'energia elàstica i distribuir lleugerament millor la deformació.

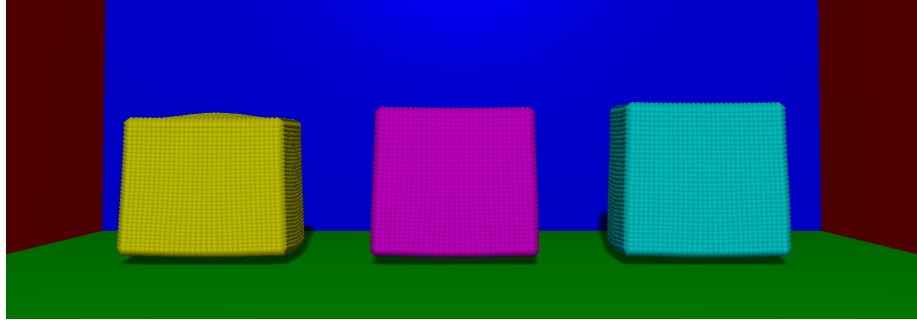


Figura 7.3: *Comparació de deformació*. Tres objectes amb diferents mòduls de Young després d'impactar contra el terra. D'esquerra a dreta: 300, 800, 1300 (valors del mòdul de Young). Simulació *cmp1*, apèndix B.5.

Aquest fet es pot observar en les simulacions *C7* i *C8* als apèndixs B.43 i B.44, on la barra només retorna en el model corrotacional.

Encara es veu més clar en les simulacions *superDeff* i *superDeffCor*, als apèndixs B.47 i B.48, on el model *Neo-Hookean* no pot mantenir l'estructura inicial de l'esfera i aquesta es trenca, mentre que la simulació que usa el corrotacional se'n surt millor en mantenir la uniformitat.

7.2.2. Deformació d'objectes

Donat un objecte a simular, el *software* és capaç d'emular les propietats de deformació d'aquest satisfactòriament, explicades a la secció 2.5.

Podem veure primer com, depenent dels coeficients del material, es pot definir el comportament d'un sòlid perquè només es deformi fins a cert punt.

El mòdul de Young defineix l'energia necessària per produir una deformació, tal com s'observa en la figura 7.3, de manera que aquest valor defineix la resistència del material a qualsevol deformació.

D'aquesta manera es produeixen les deformacions plàstiques anteriors, on l'energia ha estat absorbida pel material, provocant una deformació permanent.

Com més elevat és el mòdul de Young menys deformació es produirà, i per tant l'energia absorbida amb un impacte, emmagatzemada en el gradient de deformació elàstic de les partícules del cos, es retornarà a causa a l'elasticitat d'aquest, amb una força que es propaga a la resta de partícules gràcies a la integració de la graella.

Aquest fenomen és visible a la figura 7.4.

Per altra banda tenim el coeficient de *Poasson*, ν , que també defineix el comportament de deformació de l'objecte, concretament la proporció entre la deformació uniaxial i la tangencial.

Això es pot observar altre cop a la figura 7.3, on la base dels cubs és més ampla que la part superior.

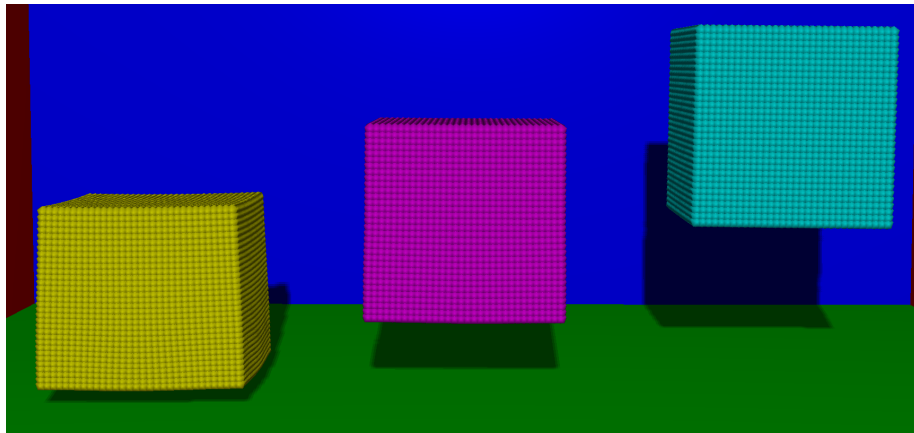


Figura 7.4: *Energia elàstica: rebot*. Tres objectes amb diferents mòduls de Young rebotant contra el terra, en el seu punt més alt. D'esquerra a dreta: 800, 5000, 10000 (valors del mòdul de Young). Simulació *cmp2*, apèndix B.6.

Les deformacions poden ser causades per molts fenòmens diferents, però sempre a partir d'una força que afecta el cos en algun sentit. Per observar els efectes d'aquest tipus d'interacció, es pot veure com una barra horitzontal fixada es deforma d'acord amb les forces de gravetat, veure figura 7.5.

7.2.3. Trencament

Quan una deformació és massa gran, i el cos ja no és capaç d'absorbir més energia, en comptes de deformar-se més aquest es trenca. El trencament ve definit pels paràmetres p_c i p_s , vists en la secció 5.2.4 en l'apartat de plasticitat.

Aquestes ruptures succeeixen en impactes importants, i/o col·lisions perpendiculars, resultant en una porció de l'objecte sent separada, com s'observa a la figura 7.6.

Encara que aquest trencament es pugui veure com dues parts de l'objecte sent separades, s'ha d'entendre a escala de partícula, de manera que si una partícula es deforma cap a una direcció per algun motiu, i la magnitud d'aquesta deformació és elevada, la partícula comença a actuar independent del cos on pertanyia, doncs no té deformació elàstica per retornar-hi.

Si apliquem aquest comportament a totes les partícules d'un cos, podem fer que aquest actuï com un líquid o fluid molt dens.

Aquest fenomen és difícil de configurar, ja que es basa en un trencament no controlable directament, però el podem forçar especificant un coeficient ν elevat, de manera que les deformacions són exagerades, i per tant les ruptures més senzilles.

Podem observar aquest fet en la figura 7.7, de manera que es produeix la liquació d'un cub fins a cert punt.

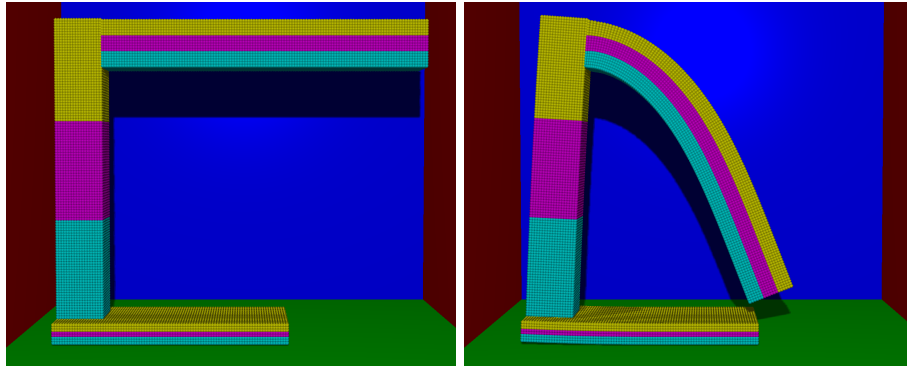


Figura 7.5: *Deformació per gravetat*. Simulació de l'estructura 'C', on l'extrem superior és d'un material tou, mentre que el tronc d'aquesta és més rígid. La primera imatge correspon a la posició original de l'estructura; la segona imatge correspon al punt de deformació màxima, només sent afectat per la gravetat. Simulació C6, apèndix B.42.

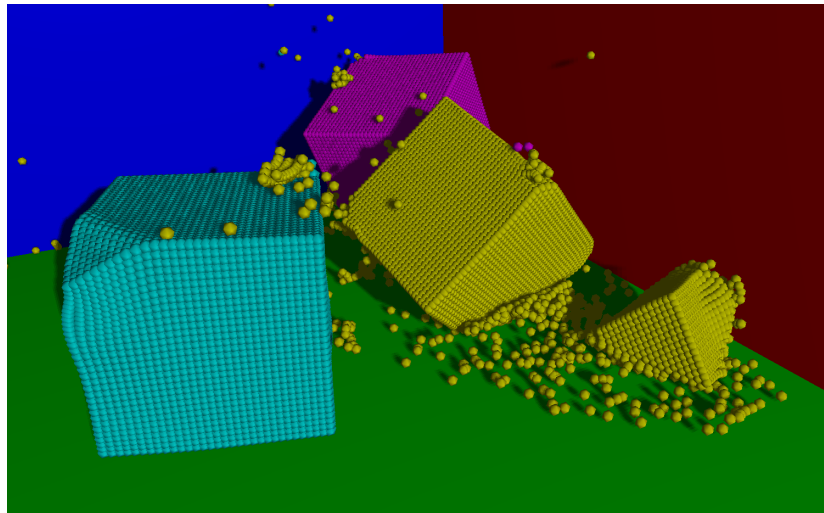


Figura 7.6: *Trencament i deformació del material*. Tres objectes idèntics després de l'impacte contra una superfície des de diferents distàncies. Ruptura produïda per les forces en l'impacte entre el cos groc i blau. Simulació prog3, apèndix B.10.

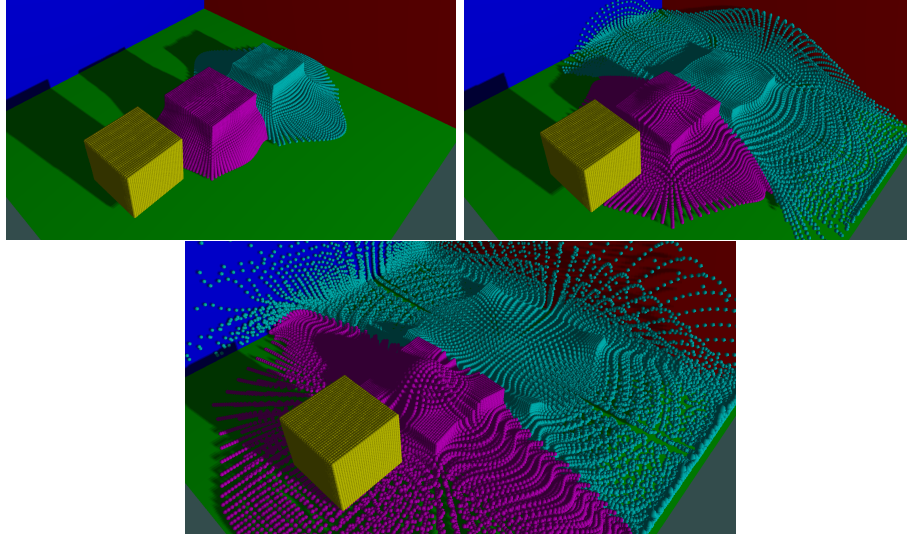


Figura 7.7: *Ruptura líquida*. Forçant el paràmetre ν , es provoca ruptura plàstica a escala de partícula, que permet emular un fluid dens. El paràmetre ν és, d'esquerra a dreta, 0.1, 0.35 i 0.45. Notar com no tot el cub ha estat liquat, i les cantonades resisteixen el trencament, doncs la deformació ha arribat amb menys intensitat en aquestes zones. Simulació *cmp4*, apèndix B.7.

7.2.4. Elasticitat

Per acabar de veure com es comporta la part elàstica d'un cos, més difícil de percebre, podem crear un material sense component plàstica de manera que tota deformació serà elàstica, i per tant tindrà forces de recuperació.

En aquest cas, el cos mai es trencarà i buscarà retornar a la seva forma original en tot moment.

Aquest fenomen es pot apreciar a les imatges de la figura 7.8, on es veu la deformació, recuperació, i impuls que el cos agafa gràcies al rebot.

És important denotar que, sobretot en el cas particular d'elasticitat completa, computar la descomposició en valors singulars d'un gradient de deformació gran és costós, i per tant simulacions d'aquestes característiques requeriran més temps per a ser completades.

7.2.5. Enduriment

Recordem, de la secció 2.5.4, que l'enduriment s'ha definit com la propietat del material a resultar més rígid davant la compressió, i tou en descompressió.

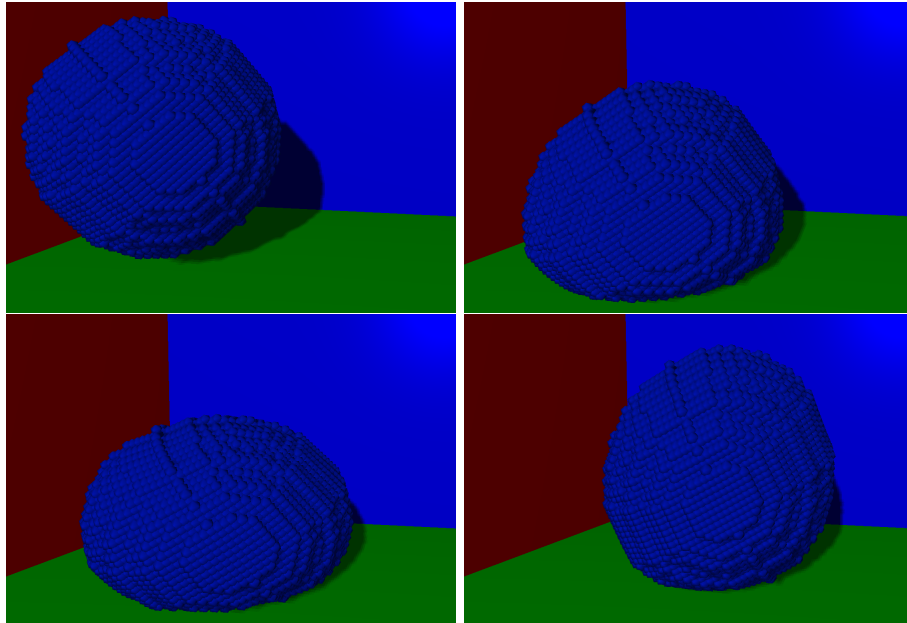


Figura 7.8: *Rebot completament elàstic*. Esfera completament elàstica, rebotant contra una superfície. Notar la deformació en l'impacte en la progressió d'imatges, d'esquerra a dreta i de dalt a baix. Simulació *nPlastic6*, apèndix B.15.

Aquest esdeveniment ja es pot observar en l'anterior figura 7.7, en el centre dels cubs, on a causa de la pressió s'ha format una estructura més sòlida i rígida. Important destacar que aquest fet es produeix només sobre deformacions plàstiques.

7.2.6. Físiques

La interacció amb elements físics externs funciona adequadament, però amb poca precisió atès que es du a terme durant el processament de la graella. Particularment, quan una cel·la està influïda per un objecte a col·lidir, potser una de les cel·les adjacents no ho està, i quan una partícula recupera la seva velocitat, aquesta només estarà parcialment afectada per la col·lisió.

Tot i això, els resultats són visualment correctes, i s'aprecia l'objecte a col·lidir en tot moment.

En la figura 7.9 es pot veure una simulació que recorda a un rellotge de sorra. Notar la deformació plàstica superior, i la ruptura progressiva en el coll del "rellotge". També es pot observar com la interacció física no és perfecta, i hi ha partícules sobresortint.

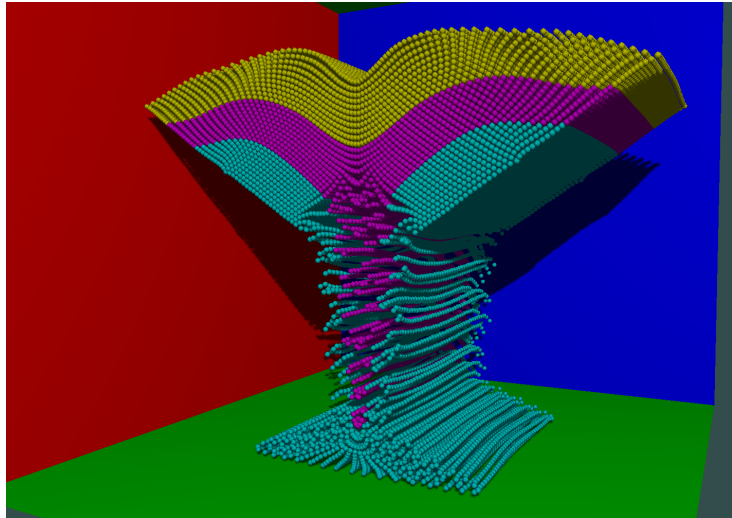


Figura 7.9: *Interacció física, rellotge de sorra*. Material deformable plàstic, a tall de fluid dens, relliscant sobre unes rampes laterals implementades amb físiques a graella. Simulació *hoursand1*, apèndix B.20.

7.2.7. Líquids

Tal com s'ha mostrat a la secció 7.2.3, podem modificar els paràmetres del material per a forçar que es comporti com un fluid, de manera que totes les deformacions segueixen el model elàstic, però són finalment plàstiques.

A la figura 7.10 es pot observar com en l'impacte de dues esferes, d'un material fàcilment líquuable, aquestes es deformen per a comportar-se com un fluid. Notar a l'última imatge que no tota l'esfera s'ha líquuat, i que petites porcions de material s'han mantingut.

7.2.8. Acoblament

Tot sòlid simulat alhora en la mateixa graella, sofrirà acoblament. És a dir, que quan dos cossos s'aproximin molt, la integració de les forces els tractarà com a un de sol.

Aquest fet es pot veure reflectit en qualsevol simulació amb impacte d'elements plàstics amb ruptura, on un element arriba a trencar a l'altre, o les partícules recobreixen el complementari.

Particularment, a la figura 7.11 s'aprecien ambdós casos.

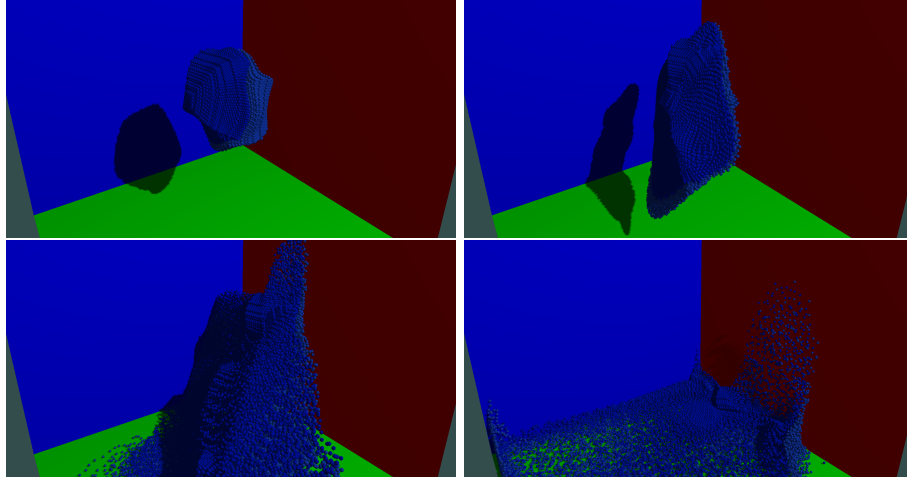


Figura 7.10: *Col·lisió i liquació de dues esferes*. Dues esferes forçades a trencar-se, $\nu = 0.39$, col·lidint entre elles a una velocitat elevada. Evolució de la interacció, d'esquerra a dreta i de dalt a baix. Simulació *snow2*, apèndix B.24

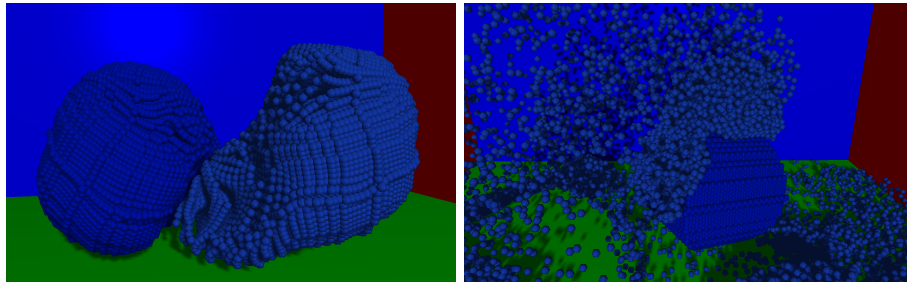


Figura 7.11: *Efectes de l'acoblament*. Primera imatge, ruptura en la distensió entre dos objectes plàstics per acoblament de les partícules a la graella. Segona imatge, partícules d'un segon objecte acoblades al primer, unides al seu cos. Simulacions *hard2* i *snow2.2*, apèndixs B.33 i B.25 respectivament.

L'acoblament és inevitable quan es discretitzen les partícules en una mateixa graella, perquè ambdós cossos poden compartir cel·les de la mateixa en la respectiva discretització.

La solució seria usar graelles independents, i calcular les interaccions entre ambdós objectes per separat, usant la graella complementària a l'hora de dur a terme la interacció.

8. Gestió del projecte

8.1. Descripció de tasques

A continuació es descriuen les diferents tasques a dutes a terme en el projecte, així com la seva definició inicial i l'evolució que han sofert durant les diferents etapes del projecte.

Base del projecte i entorn de treball

La primera tasca del projecte consisteix en la creació de les eines més bàsiques i elementals, i la familiarització amb aquestes, que han estat necessàries per a acompanyar la implementació del simulador.

Les eines a programar han consistit en, usant la llibreria gràfica *OpenGL* [31], un visualitzador per pantalla de partícules per a veure en temps real els resultats; i seguidament una eina per exportar els fotogrames en imatges, usant la llibreria *open source stb_image* [32].

Simulador de partícules simple

Fase que ha tingut com a objectiu principal l'aprenentatge del funcionament dels simuladors *MLS-MPM*, implementant la seva versió més simple a sobre de la base creada a l'anterior etapa.

Aquest primer simulador és molt semblant al codi de *2D-MPM* en 88 línies de [8], però portat al terreny del projecte.

És important denotar que aquesta fase ha estat dividida en dos: primerament s'ha dut a terme un aprenentatge exhaustiu de la nova temàtica, i posteriorment l'anomenada implementació.

Addició de model elàstic

De la mateixa manera que la fase anterior, aquesta ha estat dividida en aprenentatge i implementació, però dels models elàstics a incorporar en el simulador. En concret s'ha estudiat tant el model corrotacional fix aplicat a [8], basat en [7].

Més endavant s'ha decidit implementar i estudiar també el model *Neo-Hookean*, degut a la seva rapidesa, compatibilitat amb *APIC* [18] i per comparar resultats.

Optimització i estudi 2D

Juntament amb el director, s'ha analitzat el simulador actual i s'ha estudiat quines zones requerien alguna millora, o podien admetre optimitzacions interessants. També s'ha discutit sobre com plantejar-se la migració del simulador de dues dimensions a una tercera.

Durant aquesta etapa també es s'han valorat les oportunitats de paral·lelitzar el codi i millorar-ne la precisió del simulador, juntament amb la seva posterior implementació.

Una de les idees descartades en aquesta etapa és l'ús de *compute shaders*, per traspasar el còmput de la simulació de la CPU a la GPU.

Primera documentació

L'inici de la documentació, planificació, definició de l'abast i anàlisi detallat de l'enfocament del projecte s'ha dut a terme en aquesta tasca, a la tercera part del desenvolupament del projecte, per a puntualitzar el procediment a seguir per al seu correcte desenvolupament.

Millora a 3D

Aquesta etapa ha tractat íntegrament la translació del projecte a la nova dimensió, a discutir el procediment amb el director del projecte arribat el moment, doncs hi ha moltes maneres diferents d'aproximar aquest canvi depenent del desenvolupament del treball.

De totes formes, podem enumerar les característiques que s'han hagut de canviar concretament:

- Estructures de dades: s'ha incorporat la nova coordenada a emmagatzemar per la posició de les partícules, i modificant la graella perquè sigui tridimensional.
- Model elàstic: La matemàtica darrere el model requeria canvis importants per a poder funcionar.
- Algorisme: Per a adaptar els canvis anteriors, múltiples retocs a l'algorisme també són necessaris.
- Visualitzador: S'ha hagut d'idear alguna forma de poder visualitzar una zona de l'espai 3D de la simulació.

Per altra banda, de manera no planificada, s'ha hagut d'afegir el sistema d'emmagatzematge de simulacions, a causa de l'alt temps de computació d'aquestes i a la mancança d'informació en l'exportació en imatges.

Estudi 3D

Amb el simulador correctament implementat, s'ha pogut passar a l'etapa de documentació, estudi, i ajustament dels paràmetres del simulador. Durant aquesta fase s'han documentat totes les modificacions finals dutes a terme durant el canvi a la tercera dimensió, i perfeccionar la configuració del simulador per a obtenir resultats adequats.

Durant aquesta etapa s'ha tingut accés, de manera gratuïta, a un servidor de còmput per a dur a terme diverses simulacions en paral·lel en qualsevol moment. Aquest detall ha estat molt útil per poder fer el gran nombre de simulacions dutes a terme.

Físiques

Diferenciant l'aprenentatge de la implementació, s'ha buscat arribar a codificar un sistema de col·lisions entre objectes externs al simulador i les partícules d'aquest, de manera que es puguin dur a terme animacions amb interaccions externes.

Estava planificat implementar un sistema de càrrega de models amb malles de triangles, i posteriorment un mecanisme d'interacció i transmissió de forces unidireccional (del model al simulador), com els mostrats a [19].

Finalment només s'ha implementat la interacció amb models immòbils i implementats directament en codi, per motius temporals.

Documentació final

Finalitzant el projecte, s'ha hagut de consolidar la documentació i la memòria d'acord amb el software creat, i tancat ja per a aquesta etapa. Aquí s'han sintetitzat tots els resultats del programari, els estudis i descobertes interessants. Donat que el treball s'havia focalitzat massa com a document introductori a *MPM*, aquesta memòria ha estat fortament reestructurada. Fet que ha suposat una inversió de temps major que l'esperada.

8.2. Gestió econòmica del projecte

En aquest apartat s'analitzarà l'impacte econòmic d'aquest projecte. Com que es tracta d'una activitat de recerca i investigació no hi ha costos de producció, ni de comercialització. És a dir: es tractaran els costos del disseny, planificació, estudi, implementació, recursos humans i materials de la indagació en nous coneixements.

Per a calcular el cost d'amortització, es tindrà en compte la vida útil del component en comparació amb la duració del projecte, de fins a 6 mesos en l'actual cas.

Cal afegir que s'usarà el punt com a indicador de decimals en aquest apartat, i tres xifres decimals significatives en els càlculs.

8.2.1. Partides

El desglossament del cost econòmic del projecte es troba distribuït en les següents partides, ja incloent les desviacions temporals i econòmiques que el projecte ha patit.

Hardware

Tenim en compte el cost de dos ordinadors de sobretaula, que han servit per al desenvolupament i la renderització de simulacions. En concret s'ha disposat de les següents característiques.

- **Ordinador Sobretaula:** *AMD Ryzen 5 2600 3.40GHz x 6, GeForce GTX 1060 6GB VRAM, 16GB de RAM, 256GB SSD i 2TB HDD.*
- **Ordinador Renderitzador:** *Intel Core i7-6700 3.40GHz x 8, GeForce GTX 1050 4GB VRAM, 32GB RAM, 512GB SSD i 4TB HDD.* Cal puntualitzar que considerem que aquest PC té una vida útil inferior a causa d'estar constantment sotmès a execucions que requereixen considerable potència de manera molt continuada.
- **Servidor:** Durant el desenvolupament del projecte, s'ha aconseguit accés gratuït a un servidor de comput, que no es tindrà en compte en el preu final.

Costs representats a la taula 8.1, comptabilitzant una mitjana de 5 hores de dedicació diàries de 220 dies laborables anuals, usant la formula 8.1. Per a la comptabilització de les hores, s'estima que un 65% del treball s'ha dut a terme amb l'ordinador de sobretaula, i el restant 35% amb el renderitzador, sobre les 600 hores aproximades de duració del projecte.

$$Cost = Cost \text{ equip} \cdot \frac{1 \text{ vida útil}}{4 \text{ anys}} \cdot \frac{1 \text{ any}}{220 \text{ dies lab.}} \cdot \frac{1 \text{ dia}}{5 \text{ Hores dedicacio}} \cdot \text{Hores dedicades} \quad (8.1)$$

Producte	Cost	Vida útil	Hores	Amortització
Ordinador Sobretaula	890.000€	4 anys	390	78.886€
Ordinador Renderitzador	1 400.000€	4 anys	210	66.818€
Total	-	-	600	145.704€

Taula 8.1: *Costs Hardware.* (Elaboració pròpia)

Software

Per aquest projecte, s'ha usat un cert nombre de programes de pagament, i altres de gratuïts o *Open Source*. Els productes de pagament, han sigut obtinguts de manera gratuïta gràcies a llicències d'estudiants oferides per la FIB. Queden recollits en la taula 8.2, a continuació.

Producte	Preu	Amortització
Git	0.000€	0.000€
GitKraken ¹	48.000€	0.000€
L ^A T _E X	0.000€	0.000€
TeamGantt	0.000€	0.000€
Ubuntu OS	0.000€	0.000€
Visual Studio 2019 ²	641.000€	0.000€
Windows 10 ²	145.000€	0.000€
Total	834.000€	0.00€

Taula 8.2: *Costs Software.* (Elaboració pròpia)

Recursos Humans

Els costs de recursos humans fan referència a les diverses funcions que realitza, en aquest cas, l'únic treballador que realitza el projecte. S'ha decidit excloure del pressupost al director del projecte, ja que s'ha considerat que és tracta d'un cost de consultoria i suport gratuïts per part de la universitat.

En aquest cas el treballador ha fet les funcions de coordinador, analista, desenvolupador i tester (conegut com a *Test engineer*); i la seva dedicació en hores, així com el cost associat en cadascuna de les seves respectives tasques, es troba en les taules 8.3 i 8.4.

Els costs són estimats suposant treballadors assalariats, acollint-nos a una jornada laboral màxima de 1800 hores anuals, d'acord amb el BOE 2016-2856[35], i seleccionant els sous de la web *Indeed*[36] com a referents per als següents càlculs³. Per a calcular el cost amb la despesa de Seguretat Social, s'adjunta aquesta quantitat comptabilitzada com el 35 per cent del sou brut.

Posició	Salari brut (€/hora)	Hores	Cost	+ Seguretat Social
Coordinador	23.773	137	3 256.901€	4 396.816€
Analista	33.673	50	1 683.650€	2 272.927€
Desenvolupador	46.943	358	16 805.594€	22 687.551€
Tester	50.799	68	3 454.332€	4 663.348€
Total	-	613	25 200.477€	34 020.644€

Taula 8.3: *Cost Recursos Humans*. (Elaboració pròpia)

Costs indirectes

Els costs indirectes, donat que no són calculables precisament, s'intenten aproximar el millor possible en aquest apartat.

Pel que fa als costs elèctrics, es comptabilitza el preu per quilowatt hora a 0.11 €/kWh, i s'ha calculat un consum mitjà de l'ordinador de sobretaula de 240 Watt. Com que aquest terminal s'ha usat, aproximadament, 500 hores, podem calcular l'energia consumida com a 120 kW, o 13.2€.

Per altra banda, disposem de l'ordinador per a computar els tests. Aquest, tot i que s'usarà una mitjana de 70 hores, té un consum molt més elevat que l'ordinador de sobretaula, ja que estarà treballant a màxima potència tota l'estona. Aquest fet ens dona una potència esperada de 600 Watt, el que ens equival a una energia consumida de 42 kWh, o 4.62 €

També podem sumar el cost elèctric dels perifèrics, com les pantalles, amb un consum equivalent a 28 Watts durant 530 hores; és a dir: 14.84 kWh o 1.632€.

Afegim també els costs de transport per anar al CRV, on se situa el laboratori del ViRVIG, que es correspon amb viatges durant 3 mesos i es comptabilitzen amb els seus respectius descomptes, equivalent a 52.5€ en total.

¹Obtingut amb la llicència de *GitHub Student Pack*

²Obtingut amb la llicència de *Windows Azure Education*

³Basats en la mitjana americana, d'una població de com a mínim 3500 individus.

Id	Activitat	Hores	RRHH	Cost (€)
0	Total	613	-	25 200.477€
1	Creació de la base	25	Desenvolupador	1 173.575€
2	Simulador de partícules simple	100	-	4 732.860€
2.1	Aprenentatge simulador	65	Desenvolupador	3 051.295€
2.2	Implementació simulador	25	Desenvolupador	1 173.575€
2.3	Validació simulador	10	Tester	507.990€
3	Addició model elàstic	98	-	4 631.209€
3.1	Aprenentatge models elàstics	60	Desenvolupador	2 816.580€
3.2	Implementació models elàstics	30	Desenvolupador	1 408.290€
3.3	Validació models elàstics	8	Tester	406.339€
4	Optimització i estudi 2D	45	-	1 714.335€
4.1	Estudi 2D	30	Analista	1 010.190€
4.2	Paral·lelisme	15	Desenvolupador	704.145€
5	Primera documentació	65	-	1 545.245€
5.1	Documentació: planificació	20	Coordinador	475.46€
5.2	Documentació: redacció	35	Coordinador	832.055€
5.3	Documentació: correcció	10	Coordinador	237.730€
6	Millora a 3D	60	-	2932.260€
6.1	3D: migració	30	Desenvolupador	1 408.290€
6.2	3D: validació	30	Tester	1 523.970 €
7	Estudi i recerca 3D	60	-	2 319.48€
7.1	Estudi 3D	20	Analista	673.46€
7.2	Documentació 3D	10	Coordinador	237.730€
7.3	Estudi3D: millores i validació	30	Desenvolupador	1 408.290€
8	Físiques	80	-	3 832.560€
8.1	Aprenentatge físiques	20	Desenvolupador	938.860€
8.2	Implementació físiques	30	Desenvolupador	1 408.290€
8.3	Validació físiques	20	Tester	1 015.980€
8.4	Implementació eines	10	Desenvolupador	469.430€
9	Documentació final	80	-	1 901.840€
9.1	Documentació programa	18	Desenvolupador	844.974€
9.2	Redacció memòria	30	Coordinador	713.190€
9.3	Revisió i correcció	12	Coordinador	285.276€
9.4	Preparació i defensa final	20	Coordinador	475.460€

Taula 8.4: Cost recursos humans per tasca. (Elaboració pròpia)

I finalment cobrim els costos de connexió a la xarxa, segons l'import mensual de 36.99€, durant els 6 mesos de duració del projecte, equivalent en definitiva 221.94€.

A la taula 8.5 podem observar un resum.

Font	Cost(€)
Ordinador sobretaula	13.200€
Ordinador rendering	4.620€
Perifèrics	1.632€
Transport	52.500€
Connexió a internet	221.940€
Total	293.892€

Taula 8.5: *Costs Indirectes*. (Elaboració pròpia)

8.2.2. Pressupost final

Finalment, reunim en la següent taula 8.6 el sumatori de totes les partides del pressupost, per a obtenir una visió general del cost del projecte i la distribució de la inversió teòrica en aquest.

Font	Cost(€)
Hardware	145.704€
Software	0.000€
Recursos Humans	34 020.644€
Costs indirectes	293.892€
Total	34 640.240€

Taula 8.6: *Pressupost final*. (Elaboració pròpia)

8.2.3. Viabilitat econòmica

Aquest projecte és d'investigació, recerca i divulgació; i per tant no es cerca obtenir beneficis. D'aquesta manera, la viabilitat econòmica depèn de les subvencions i ajuts rebuts per part dels contribuents, que fan possible la realització de molts projectes d'aquesta temàtica i que segueixen la mateixa noció.

De totes maneres, la viabilitat econòmica aniria estrictament lligada a un control ajustat del pressupost, seguint uns certs procediments al final de cadascuna de les tasques a realitzar. L'objectiu d'aquest seguiment és calcular la diferència entre el cost esperat i el real, detectar desviacions en el pressupost i corregir-les com més aviat millor.

8.3. Informe de Sostenibilitat

La sostenibilitat no és una qüestió en la qual es pensi activament ni de bon tros, però és inherent a qualsevol activitat que es du a terme, concepte, producte, formació... i particularment en tot el que involucra recursos TIC, que en l'era de la informació són omnipresents.

Trobo essencial tenir intenció; intenció d'aprendre i millorar el nostre entorn, intenció de ser crític, i intenció de fer les coses bé. Pot ser que un mai hagi estat instruït en tècniques, conceptes, o indicadors de sostenibilitat; però si es té intenció de valorar i enriquir el tarannà de si mateix, molt errat no es pot anar. Amb això vull dir que no és necessari ser un expert per a poder aportar quelcom a l'entorn i la societat, i a ajudar al fet que aquesta sigui més sostenible.

Aquesta intenció també demostra un interès actiu i una consciència de l'impacte que un mateix pot arribar a tenir, reflectida en no només el "què" es fa, sinó amb el "com" es fa. Aquesta és la meua mentalitat al dur a terme aquest projecte.

La comprensió del poder que es té amb els recursos TIC ens atorga la capacitat, inevitablement, d'afectar en major o menor mesura el nostre entorn i, per tant, si busquem un àmbit més sostenible, estarem maximitzant l'impacte positiu de la nostra activitat professional sobre la societat.

Com a reflexió cal dir que és necessari tenir intenció, però no és suficient; trobo que a vegades cal anar més enllà, ser crític amb un mateix, i intentar millorar el que ja s'ha fet, cercar informació i aprendre per a fer una millor feina, i ser més sostenible. Un exemple clar seria l'aprenentatge i comprensió d'indicadors de les tres dimensions de la sostenibilitat, que ajuden a la correcta mesura i apreciació del potencial impacte del projecte.

Aquest fet es transmet a escala personal, tot i sempre intentar dur a terme les coses de manera correcta, he de reconèixer que mesurar a l'hora de la veritat la potencial empremta del projecte és una tasca desconeguda per a mi, així com el control de la dimensió econòmica i el coneixement d'indicadors de sostenibilitat. Per això parlo d'intenció, perquè espero millorar en aquest aspecte, i aconseguir elevar el treball a un següent nivell.

8.3.1. Dimensions de la sostenibilitat

Dimensió econòmica

Com s'ha mencionat anteriorment a la secció 8.2.3, el projecte té un cost econòmic teòric estimat de 34 460.240€. Tot i això, la viabilitat econòmica del projecte posat en producció no és valorable en beneficis monetaris, doncs aquests no es produiran. La motivació del projecte, a més de personal, busca millorar les solucions existents, i conseqüentment ha d'animar a altres investigadors a millorar l'estat actual de *MPM* i dur a terme millores progressives; fet que provocarà l'aparició de nous projectes mentre aquest sigui una solució acceptable i potent.

És un projecte de recerca i investigació amb un cost moderadament elevat, a causa de l'amplitud de coneixements a incorporar i a validar, però en futurs projectes aquest cost serà amortitzat, ja que aquests coneixements ja estaran adquirits.

En aquest projecte, concretament, busquem una solució més eficient que permeti simular més ràpidament un entorn concret, i sense necessitar un gran nombre de recursos; disminuint el desgast de components d'un processador, i reduint el cost econòmic de temps de simulació necessaris.

Dimensió ambiental

Els costos ambientals durant la realització del projecte, poden ser mesurats amb les dades obtingudes a la secció 8.2.1 en quilowatts hora consumits, d'un total de 176.67kWh de consum energètic durant l'etapa del projecte posat en producció (PPP). Caldria comptabilitzar, a més a més, les emissions de CO₂ produïdes pel transport públic usat per a viatjar al ViRVIG, i produïdes per generar l'electricitat consumida.

En el pitjor dels supòsits, en que la totalitat de l'electricitat prové d'una planta de crema de carbó, podem calcular una producció de 0.94Kg de CO₂ per cada quilowatt hora consumit; és a dir: 166.067Kg de CO₂ produïts pel projecte. Pel que fa al transport, sempre s'ha viatjat en tren o autobús, i s'ha calculat una aproximació 8.16Kg de CO₂ produït per aquest, donada una producció de 0.98 grams de CO₂ per quilòmetre. Per tant, hem generat una petjada de carboni equivalent a 174.227Kg de CO₂.

Afegim també 70Kg de CO₂ extra, per comptabilitzar d'alguna manera l'impacte de la despesa energètica del servidor emprat en aquest projecte.

Pel que fa a la vida útil del projecte, des d'un punt de vista ambiental pot aportar una millora, ja que aquest busca reduir, i minimitzar, el temps i l'energia requerida per a efectuar certes simulacions. Això recau en un consum elèctric menor, i per tant una menor petjada de carboni respecte a les solucions existents actualment.

Pel que fa a riscos de caràcter ambiental, aquest projecte no en provoca de cap classe. En el pitjor dels casos, no es produiria cap millora, i ens quedaríem en l'estat previ a l'inici del projecte; i en el millor dels casos, haurem millorat les solucions existents de manera més sostenible.

Dimensió social

Aquest projecte ha proporcionat a l'autor una àmplia base en simulació de fluids, així com en disciplines molt diferenciades de la branca canònica de la informàtica. L'aprenentatge d'aquesta temàtica ha sorgit d'un interès personal, el qual es demostrarà útil i una experiència productiva per a futurs projectes de tota mena, així com estimular la reusabilitat del mateix treball per a aprofundir més en aquest.

En l'àmbit social, aquest projecte és difícil que millori de manera significativa l'estat de l'art actual dels simuladors de fluids; però sí que busca ser un punt de partida i introducció en aquest extens i difícil camp, ja que existeixen pocs documents a la literatura, o a la comunitat, que puguin ser usats per qualsevol amb l'objectiu d'introduir-se en la simulació de fluids, d'aquesta manera s'espera que aquesta memòria pugui servir per a suplir aquesta mancança.

Si aquest últim punt és ben aprofitat, les possibilitats són molt altes que s'usi aquest treball com a base per a que altres estudiants puguin reproduir els resultats aquí descrits, i aprofundir en altres subcampus de la simulació de fluids.

En contrast, la possibilitat que el projecte tingui un impacte negatiu a la societat és molt baixa, i depèn de la qualitat del treball per a servir de guia futura a altres alumnes, o persones que vulguin aprendre sobre aquesta temàtica.

8.3.2. Matriu de sostenibilitat

A manera de resum, presentem la següent matriu de sostenibilitat representada a la taula 8.7 per a sintetitzar els anteriors apartats.

	PPP	Vida útil	Riscs
Econòmica	38 917.670€	No beneficis	La no viabilitat econòmica
Ambiental	244.227KgCO ₂	Reducció energètica	La no millora
Social	9/10	7/10	Pocs

Taula 8.7: *Matriu de sostenibilitat*. (Elaboració pròpia)

9. Conclusions

En aquest treball de final de grau s'han presentat tots aquells coneixements necessaris per entendre, desenvolupar, i implementar un simulador qualsevol basat en *MPM*, seguint la derivació de *Navier-Stokes*.

Per consolidar aquests coneixements, s'ha implementat un simulador híbrid de *MPM-MLS*, que permet simular diferents materials de manera molt configurable.

Amb aquest, veiem la quantitat de possibilitats de simulacions que es poden obtenir, configurant adequadament els paràmetres, a partir d'un mateix algoritme base.

Havent vist com es modelen les deformacions, tant elàstiques com plàstiques, es poden idear diferents combinacions, i comprendre com aprofitar-les per obtenir aquests diferents fenòmens com són la liquació, o l'enduriment.

D'altra banda, també s'ha tractat com la interacció amb físiques funciona sense afegir grans sobre costos en l'algorisme, i aporta una eina senzilla per influir molt potentment en les simulacions.

També s'ha implementat un visualitzador de simulacions, juntament amb les eines necessàries per a emmagatzemar i recuperar aquestes en el format d'arxius propi *sbfi*, o en format *gif*.

La implementació intenta ser modular i configurable, de manera que no només es permet dur a terme un ampli ventall d'operacions, sinó que el codi resulta senzill de llegir, i d'entendre.

En l'àmbit personal aquest treball ha estat un gran repte, ja que m'ha obligat a submergir-me en un món allunyat de tots els coneixements vists fins ara tant en el grau d'Enginyeria Informàtica, com en els meus projectes propis, i en el que resulta difícil entrar a causa de la tecnicitat d'aquest. Així i tot, aquest projecte em generava un gran interès i curiositat, i sens dubte ha resultat una experiència fructuosa.

M'ha fet redescobrir les matemàtiques des d'un altre punt de vista, i el seu lligam amb les ciències de computadors, i motivat a seguir treballant en aquest camp.

Per aquest motiu espero que altres estudiants puguin usar aquest mateix treball per també introduir-se en la simulació de materials deformables, així com conèixer les possibilitats que aquesta temàtica obre.

No obstant això, encara hi ha marge de millora, tant en el treball escrit com en la implementació del simulador, doncs amb més dedicació s'està convençut de que els resultats mostrats en aquest treball podrien ser millors i més variats.

9.1. Competències tècniques

Aquest treball de final de grau ha estat dut a terme seguint les següents competències tècniques, les quals aquí s'expliquen i justifiquen.

CCO2.1: Demostrar coneixement dels fonaments, dels paradigmes i de les tècniques pròpies dels sistemes intel·ligents.

Un simulador de materials deformables és una aplicació que s'enfronta a la interpretació d'un espai dinàmic, i amb comportament incert a priori. D'aquesta manera busca poder computar tot un seguit d'interaccions per inferir l'estat del sistema en el futur, interactuant amb el cos a simular i el seu entorn.

Per això s'usen tècniques de discretització i transformació de dades, per a portar el problema a un terreny tractable i poder trobar la millor aproximació possible.

CCO2.2: Capacitat per a adquirir, obtenir, formalitzar i representar el coneixement humà d'una forma computable per a la resolució de problemes.

La mateixa base d'un simulador de materials elàstics consisteix a idear, desenvolupar, i implementar la representació d'un material en diferents descripcions (*Euleriana* i *Lagrangiana*), i a partir d'aquestes representar la deformació, tant plàstica com elàstica, el trencament, i les forces que l'afecten.

CCO2.6: Dissenyar i implementar aplicacions gràfiques, de realitat virtual, de realitat augmentada i videojocs.

Ha estat essencial programar un visualitzador gràfic de simulacions, de manera que permeti apreciar els comportaments i propietats dels materials simulats, i introdueixi certa interacció a dins del visualitzador.

Aquest visualitzador implementa diverses tècniques de gràfics per computador per aconseguir aquests objectius.

Finalment, cal tenir en compte que el simulador implementat és un que busca obtenir resultats visualment realistes.

CCO3.1: Implementar codi crític seguint criteris de temps d'execució, eficiència i seguretat.

Tot i que no és un objectiu primari en aquest treball, ha estat necessari implementar codi eficient per a accelerar les simulacions, doncs tot petit *speedup* es fa notar davant els elevats temps de simulació.

Totes les optimitzacions dutes a terme han buscat, analitzant cada porció de l'algorisme, millorar l'eficiència d'aquest mantenint la correctesa.

CCO3.2: Programar considerant l'arquitectura hardware, tant en ensamblador com en alt nivell

En la creació de les estructures de dades, s'ha tingut en compte la seva mida per a millorar la localitat en memòria, i evitar assignacions de la mateixa dada dividida en diverses línies de *cache*.

També s'ha tingut en compte l'assignació de memòria, de manera que sempre sigui contínua, augmentant també la localitat.

9.2. Treball futur

Tot i que es considera que la part principal del projecte està acabada, el que és l'algorisme, hi ha molts detalls que no s'han pogut arribar a implementar o a estudiar, tant per falta de temps com per prioritització de tasques.

Aquestes són:

- Aprofundir en l'estudi dels diferents models d'hiperelasticitat implementats, *Neo-Hookean* i corrotacional fix, doncs són essencialment diferents però no s'ha trobat un tipus de simulació que ho mostri.
- Afegir una interfície de personalització dels models elàstics, per a poder externalitzar del simulador la definició del material, i així poder afegir-ne més, i amb major facilitat.
- Afegir un sistema de càrrega de models en malles de triangles, amb la seva voxelització, i emplenament amb partícules per a usar la *mesh* en una simulació (secció 6.1.1), o pel tractament amb físiques (secció 6.1.2).
- Físiques amb cossos en moviment, de manera que aquest moviment sigui configurable, tal com s'ha explicat en la secció 4.5.
- Simulació amb elements discretitzats en diferents graelles, de manera que els diferents cossos no s'influeixen entre ells, tal com mostra Tampubolon et al. a [29].
- Implementació del simulador amb integració implícita, de manera que sigui més ràpid dur a terme simulacions, i la realització d'aquestes sigui més segura.
- Implementació de millores en el paral·lelisme, sobretot en l'etapa de *P2G*, com s'ha explicat a la secció 5.3.1, on s'han de realitzar acurades divisions per evitar *data races*.
- Estudi i creació d'una graella basada en *SPGrid*[25], esmentada a la secció 5.3.4, per a millorar la localitat de les dades, i la rapidesa de l'algorisme.
- Renderització de les simulacions com a cossos continus, usant les densitats de la discretització a la graella, per obtenir uns resultats visualment realistes.
- Interpolació de fotogrames, per a una animació més fluida.

9.3. Agraïments

Per acabar, m'agradaria agrair a tota aquella gent que m'ha acompanyat personalment, i m'ha ajudat a seguir i a acabar aquest treball de final de grau.

Primerament, agrair al meu tutor i director del projecte Toni Susín Sánchez, per totes les seves explicacions, la motivació i implicació que ha mostrat, i per voler portar aquest treball amb mi.

Gràcies també a la meva família, en particular a la meva mare, per donar-me suport en tot moment, i per ajudar-me amb aquesta tasca titànica, per revisar el projecte, i assistir-me en tot el possible.

També, voldria agrair a tots els amics i companys que m'han escoltat, i acompanyat en aquest viatge, i acceptat la meva vida d'ermità. Gràcies per tots els bons moments i el suport que se m'ha donat.

Finalment, he de donar les gràcies al servidor d'Antoni Casas, per haver-me ajudat a minar bitcoins durant aquest projecte, i a realitzar algunes simulacions també.

Índex de figures

2.1	<i>Gradient de deformació</i>	18
2.2	<i>Vectors de tracció superficial sobre un tetraedre</i>	22
2.3	<i>Deformació plàstica</i>	29
3.1	<i>Discretització de l'espai</i>	32
3.2	<i>Funció d'interpolació, spline quadràtica</i>	34
6.1	<i>Comparació ombrejat local</i>	66
6.2	<i>Comparació shadow map</i>	67
7.1	<i>Explosió d'una simulació</i>	71
7.2	<i>Acció del model elàstic</i>	72
7.3	<i>Comparació de deformació</i>	73
7.4	<i>Energia elàstica: rebot</i>	74
7.5	<i>Deformació per gravetat</i>	75
7.6	<i>Trencament i deformació del material</i>	75
7.7	<i>Ruptura líquida</i>	76
7.8	<i>Rebot completament elàstic</i>	77
7.9	<i>Interacció física, rellotge de sorra</i>	78
7.10	<i>Col·lisió i líquidació de dues esferes</i>	79
7.11	<i>Efectes de l'acoblament</i>	79
A.1	<i>Camps vectorials</i>	100
A.2	<i>Components de la tensió</i>	105
A.3	<i>Descomposició polar del gradient de deformació</i>	107
A.4	<i>Tracció sobre el mateix cos</i>	114

Índex de taules

1.1	<i>Nomenclatura</i>	14
1.2	<i>Llistat d'abreviatures</i>	15
7.1	<i>Temps de simulació</i>	71
8.1	<i>Costs Hardware</i>	83
8.2	<i>Costs Software</i>	83
8.3	<i>Cost RRHH</i>	84
8.4	<i>Cost RRHH per tasca</i>	85
8.5	<i>Costs Indirectes</i>	86
8.6	<i>Pressupost final</i>	86
8.7	<i>Matriu de sostenibilitat</i>	89

Referències

- [1] D. Sulsky, S. J. Zhou, and H. L. Schreyer, “Application of a particle-in-cell method to solid mechanics,” *Computer Physics Communications*, vol. 87, pp. 236–252, may 1995. [11](#), [12](#)
- [2] D. Terzopoulos and K. Fleischer, “Modeling inelastic deformation: Viscoelasticity, plasticity, fracture,” *SIGGRAPH Comput. Graph.*, vol. 22, pp. 269–278, June 1988.
- [3] R. Bridson, *Fluid Simulation for Computer Graphics*. A K Peters/CRC Press, nov 2018. [12](#)
- [4] Y. Zhu and R. Bridson, “Animating sand as a fluid,” *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 965–972, 2005. [12](#)
- [5] A. Stomakhin, C. Schroeder, L. Chai, J. Teran, and A. Selle, “A material point method for snow simulation,” *ACM Transactions on Graphics*, vol. 32, no. 4, 2013. [12](#), [30](#), [38](#), [56](#)
- [6] J. Hegemann, C. Jiang, C. Schroeder, and J. M. Teran, “A level set method for ductile fracture,” in *Proceedings - SCA 2013: 12th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pp. 193–202, 2013. [12](#)
- [7] C. Jiang, C. Schroeder, J. Teran, A. Stomakhin, and A. Selle, “The material point method for simulating continuum materials,” *ACM SIGGRAPH 2016 Courses, SIGGRAPH 2016*, pp. 1–52, 2016. [12](#), [16](#), [28](#), [33](#), [80](#), [116](#), [117](#)
- [8] Y. Hu, Y. Fang, Z. Ge, Z. Qu, Y. Zhu, A. Pradhana, and C. Jiang, “A moving least squares material point method with displacement discontinuity and two-way rigid body coupling,” *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, p. 150, 2018. [12](#), [37](#), [39](#), [80](#)
- [9] Y. Hu, Y. Fang, Z. Ge, Z. Qu, Y. Zhu, A. Pradhana, and C. Jiang, “High-performance material point method (MPM) solver for Taichi. (ACM Transactions on Graphics, SIGGRAPH 2018).”
- [10] D. A. Fleisch, *A student’s guide to vectors and tensors*. Cambridge University Press, 2011. [38](#), [99](#)
- [11] “Vector field plotter — academo.org - free, interactive, education..” <https://academo.org/demos/vector-field-plotter/>. (Accessed on 02/12/2019). [100](#)
- [12] S. Dobek, “Fluid dynamics and the navier-stokes equation.” https://www.cs.umd.edu/~mount/Indep/Steven_Dobek/dobek-stable-fluid-final-2012.pdf, 5 2012. (Accessed on 12/08/2019). [102](#)
- [13] A. J. Smits, *A physical introduction to fluid mechanics*. Wiley, 2000. [102](#)

- [14] J. Bonet and R. D. Wood, *Nonlinear continuum mechanics for finite element analysis*. Cambridge university press, 1997. 16, 117
- [15] E. A. de Souza Neto, D. Peric, and D. R. Owen, *Computational methods for plasticity: theory and applications*. John Wiley & Sons, 2011. 24
- [16] M. Yuu, *Elasto-plastic Constitutive Relations*, pp. 122–153. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. 30
- [17] F. H. Harlow and M. Evans, “A machine calculation method for hydrodynamic problems,” *LAMS-1956*, 1955. 33
- [18] C. Jiang, C. Schroeder, A. Selle, J. Teran, and A. Stomakhin, “The affine particle-in-cell method,” *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 51, 2015. 35, 38, 80
- [19] C. Ericson, *Real-time collision detection*. CRC Press, 2004. 38, 82
- [20] T. F. Gast, C. Schroeder, A. Stomakhin, C. Jiang, and J. M. Teran, “Optimization integrator for large time steps,” *IEEE transactions on visualization and computer graphics*, vol. 21, no. 10, pp. 1103–1115, 2015.
- [21] “Eigen.” https://eigen.tuxfamily.org/index.php?title=Main_Page. (Accessed on 01/03/2020). 45
- [22] “Opengl mathematics.” <https://glm.g-truc.net/0.9.9/index.html>. (Accessed on 01/03/2020). 45, 65
- [23] “Openmp-api-specification-5.0.” <https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf>, 2018. (Accessed on 01/05/2020). 57
- [24] Y. Hu, Y. Fang, Z. Ge, Z. Qu, Y. Zhu, A. Pradhana, and C. Jiang, “A moving least squares material point method with displacement discontinuity and two-way rigid body coupling (supplementary document),” 57, 60
- [25] R. Setaluri, M. Aanjaneya, S. Bauer, and E. Sifakis, “Spgrid: A sparse paged grid structure applied to adaptive smoke simulation,” *ACM Trans. Graph.*, vol. 33, Nov. 2014. 60, 92
- [26] “Glfw - an opengl library.” <https://www.glfw.org/>, 2020. (Accessed on 01/06/2020). 65
- [27] “glad: Multi-language vulkan/gl/gles/egl/glx/wgl loader-generator.” <https://github.com/Dav1dde/glad>, Nov. 2019. (Accessed on 01/06/2020). 65
- [28] C. Tangora, “gif-h: Simple c++ one-header library for the creation of animated gifs from image data..” <https://github.com/charlietangora/gif-h>. (Accessed on 01/06/2020). 69
- [29] A. P. Tampubolon, T. Gast, G. Klár, C. Fu, J. Teran, C. Jiang, and K. Mu-seth, “Multi-species simulation of porous sand and water mixtures,” *ACM Trans. Graph.*, vol. 36, July 2017. 92

- [30] D. J. Anderson, *Kanban: Successful Evolutionary Change For Your Technology Business*. 2010.
- [31] Silicon Graphics and Khronos Group, “OpenGL - The Industry Standard for High Performance Graphics,” 2013. (Accessed on 28/09/2019). 80
- [32] “stb: stb single-file public domain libraries for C/C++.” <https://github.com/nothings/stb>. (Accessed on 27/09/2019). 80
- [33] S. Nelson, “LATEX: A document preparation system user’s guide and reference manual,” *Computers & Mathematics with Applications*, vol. 29, no. 11, p. 108, 1995.
- [34] Teamgantt, “Online Gantt Chart Software — TeamGantt.” <https://www.teamgantt.com/>, 2019.
- [35] M. D. Empleo Y Seguridad Social, “Disposición 2856 del BOE núm. 70 de 2016,” 2016. Capítulo IV. Artículo 22. Jornada laboral. 84
- [36] “Comparación de sueldos, buscar sueldos — Indeed.com.” <https://www.indeed.com/salaries>. (Accessed on 02/10/2019). 84

A. Teoria

A.1. Càlcul vectorial

El primer que podem usar per definir el moviment d'un fluid és un camp vectorial, perquè tècnicament podríem concretar les forces exactes que influeixen un fluid, amb un camp vectorial infinitesimal. D'aquesta manera podem representar una direcció i una magnitud que defineix el sistema a cada punt del camp. Els camps vectorials, a escala real, ens ajuden a definir, discretitzar i aproximar el moviment i forces afectant un fluid de diverses maneres.

En els següents subapartats s'explicaran i es definiran les operacions necessàries per a comprendre el contingut d'aquest document; si es vol una descripció més acurada i en profunditat, a escala d'estudiant, recomano el llibre de Fleisch [10] redactat amb estudiants com a públic objectiu.

A.1.1: Operacions bàsiques

Per a definir les operacions més elementals del càlcul vectorial usarem vectors tridimensionals $\vec{A} = (x, y, z)$, amb els que definim les següents operacions bàsiques. Per simplificació $\hat{i} = (1, 0, 0)$, $\hat{j} = (0, 1, 0)$, $\hat{k} = (0, 0, 1)$ vectors unitaris.

- Mòdul:

$$|\vec{A}| = \sqrt{A_x^2 + A_y^2 + A_z^2}$$

- Producte escalar:

$$\vec{A} \bullet \vec{B} = A_x B_x + A_y B_y + A_z B_z = |\vec{A}| |\vec{B}| \cos(\alpha)$$

- Producte vectorial:

$$\vec{A} \times \vec{B} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{vmatrix}$$
$$|\vec{A} \times \vec{B}| = |\vec{A}| |\vec{B}| \sin(\alpha)$$

Per altra banda, també es denotarà la derivada parcial de la següent manera $\frac{\partial}{\partial x}$, a diferència de les derivades ordinàries $\frac{d}{dx}$. La idea és poder definir les derivades ordinàries a través de les parcials; per exemple, donada una funció $Z = Z(x, y)$ podem definir la derivada total dZ :

$$dZ = \frac{\partial Z}{\partial x} dx + \frac{\partial Z}{\partial y} dy \quad (\text{A.1})$$

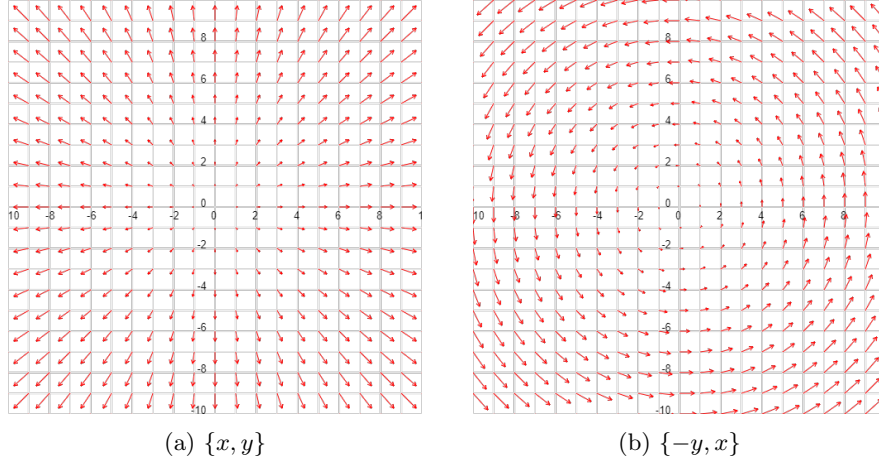


Figura A.1: Operacions sobre els camps vectorials i escalars. Camps vectorials d'exemple. (Efectuats amb [11])

A.1.2: Operador nabla $\vec{\nabla}$

Representat amb l'operador $\vec{\nabla}$, o simplement ∇ , en diferents formes, expressa l'operació vector diferencial. És a dir que pot aparèixer sol, representant la mateixa funció, o acompanyat d'una variable significat el resultat del càlcul.

$$\vec{\nabla} \equiv \hat{i} \frac{\partial}{\partial x} + \hat{j} \frac{\partial}{\partial y} + \hat{k} \frac{\partial}{\partial z} \quad (\text{A.2})$$

On recordem $\hat{i}, \hat{j}, \hat{k}$ són els vectors unitaris.

Aquest operador el podem trobar en diverses formes diferents, representant gradient ($\vec{\nabla}$), divergència ($\nabla \bullet$), rotacional (curl en anglès, $\vec{\nabla} \times$), i finalment el Laplacà (∇^2).

Per exemplificar les operacions usarem una funció escalar Φ i un camp vectorial $\vec{\Psi}$ o simplement Ψ .

Gradient

Amb l'operador $\vec{\nabla}$ seguit d'una funció escalar, que podem visualitzar com un camp, ens retorna amb quina magnitud aquest camp canvia respecte de l'espai, i la direcció del gradient indica cap a on aquest camp incrementa més pronunciadament.

$$\text{grad}(\Phi) \equiv \vec{\nabla} \Phi = \hat{i} \frac{\partial \Phi}{\partial x} + \hat{j} \frac{\partial \Phi}{\partial y} + \hat{k} \frac{\partial \Phi}{\partial z} \quad (\text{A.3})$$

D'aquesta manera representem el creixement de la funció Φ amb un altre camp vectorial. Com a exemple, el gradient de la funció $\zeta = \frac{x^2}{2} + \frac{y^2}{2}$ està representat a A.1a, on $\vec{\nabla} \zeta = \{x, y\}$

Divergència

L'operador $\nabla \bullet$ indica el còmput de la divergència d'un camp vectorial. Aquest concepte defineix la tendència dels vectors de "fluir" cap a dins o cap a fora d'un punt d'interès.

La divergència assigna un camp vectorial a un escalar, representant aquesta magnitud de flux en el camp. Tal com està definit el producte escalar, definim:

$$\text{div}(\vec{\Psi}) = \nabla \bullet \vec{\Psi} \equiv \frac{\partial \Psi_x}{\partial x} + \frac{\partial \Psi_y}{\partial y} + \frac{\partial \Psi_z}{\partial z} \quad (\text{A.4})$$

On recordem que $\vec{\Psi}$ és un camp vectorial.

Podríem entendre els valors de divergència a cada punt de l'espai, com la tendència que té el sistema d'emetre o absorbir el que representa el camp, com si es tractés d'un flux.

Rotacional

El rotacional, o *curl*, simbolitzat amb l'operador $\vec{\nabla} \times$ significa la tendència del camp vectorial a circular al voltant d'un punt concret. El resultat d'aquesta operació és un vector, de manera que té direcció i magnitud.

La direcció indica la perpendicular al pla en què la circulació del camp és màxima. La magnitud és proporcional a la circulació en el punt.

Aquesta operació és l'equivalent del producte vectorial diferencial, i només està definida correctament en la tercera dimensió com a vector doncs en dues dimensions es pot representar com un escalar (angle proporcional de rotació):

$$\vec{\nabla} \times \vec{A} = (\hat{i} \frac{\partial}{\partial x} + \hat{j} \frac{\partial}{\partial y} + \hat{k} \frac{\partial}{\partial z}) \times (\hat{i} A_x + \hat{j} A_y + \hat{k} A_z) \quad (\text{A.5})$$

$$= (\frac{\partial A_z}{\partial y} - \frac{\partial A_y}{\partial z}) \hat{i} + (\frac{\partial A_x}{\partial z} - \frac{\partial A_z}{\partial x}) \hat{j} + (\frac{\partial A_y}{\partial x} - \frac{\partial A_x}{\partial y}) \hat{k} \quad (\text{A.6})$$

En el cas d'un camp bidimensional Ψ , ens trobaríem en què només la component corresponent a \hat{k} té magnitud, ja que en 2 dimensions $\frac{\partial}{\partial z} = 0$ i $\frac{\partial A_z}{\partial z} = 0$. Per tant:

$$\vec{\nabla} \times \Psi = (\frac{\partial \Psi_y}{\partial x} - \frac{\partial \Psi_x}{\partial y}) \hat{k} \quad (\text{A.7})$$

D'aquesta manera, és senzill veure que cada component del rotacional ens indica quina és la tendència a ciclar del camp sobre cadascun dels plans perpendiculars als vectors unitaris $\hat{i}, \hat{j}, \hat{k}$.

Per il·lustrar el rotacional, podem posar com a exemple el camp bidimensional $\zeta = \{-y, x\}$, visible a [A.1b](#), del qual el seu rotacional és:

$$\vec{\nabla} \times \zeta = (\frac{\partial(x)}{\partial x} - \frac{\partial(-y)}{\partial y}) \hat{k} = 2\hat{k}$$

De manera que el camp gira en sentit contrari a les agulles del rellotge, ja que el rotacional és positiu (regla de la mà dreta).

Laplacià

El laplacià, indicat amb ∇^2 o Δ , defineix la segona derivada d'una funció escalar Φ que pot ser expressat:

$$\nabla^2 \Phi \equiv \delta \Phi = \vec{\nabla} \cdot \vec{\nabla} \Phi = \frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} + \frac{\partial^2 \Phi}{\partial z^2} \quad (\text{A.8})$$

El qual pot ser expressat com una combinació del gradient i la divergència d'aquesta funció. Aquesta operació resulta molt útil, si es té en compte que la segona derivada expressa mesures com l'acceleració.

A.1.3: Tensors

Els tensors són objectes algebraics que serveixen per fer una aplicació lineal d'un conjunt d'objectes algebraics a un altre. Aquesta definició és molt oberta, i engloba els escalars i vectors; per posar un exemple, qualsevol nombre escalar k per si sol defineix una aplicació lineal $f : \mathbb{R} \mapsto \mathbb{R}$, si definim $f(x) = kx$. Per altra banda, un producte escalar també pot ser expressat com un tensor doncs aplica linealment dos vectors a un escalar.

Els tensors es defineixen per rangs. Un tensor de rang 0 és un escalar, i un tensor de rang 1 és un array amb 3 components. Podem desenvolupar aquesta definició estipulant que un tensor de rang n és un *array* de 3^n valors. Qualsevol matriu de transformació 3x3 és un tensor de rang 2.

Concretament, en tensors de rang 2 es defineixen un seguit d'operacions especials, entre les quals potser la més usada és el producte escalar doble, definit amb l'operador $:$ entre dos tensors. Aquesta operació retorna un valor escalar respecte els elements a i b dels tensors A i B respectivament, i té dues possibles definicions:

$$A : B = \sum_i \sum_j a_{ij} b_{ji} \quad (\text{A.9})$$

$$A : B = \sum_i \sum_j a_{ij} b_{ij} \quad (\text{A.10})$$

Generalment aquesta operació es realitza sobre tensors B simètrics, de manera que és irrellevant quina de les dues definicions s'aplica; però en cas contrari, s'ha de confirmar quina definició s'usa en el determinat context.

A.2. Equació de Navier-Stokes

Introduïm les equacions de Navier-Stokes, i com s'apliquen a les simulacions de diversos fenòmens, entre els quals es troben els materials elàstics, com líquids, fum i foc. La següent explicació és una versió ampliada de la de Dobek [12]. Per una derivació molt més al detall consultar el llibre de Smits [13].

A.2.1: Camp de velocitat del flux

L'objectiu és aconseguir una descripció del moviment d'un medi continu, o una velocitat de flux (també anomenada velocitat macroscòpica), per a obtenir un camp vectorial \vec{u} que descriu la direcció i velocitat.

Aquest camp varia en funció del temps, és més correcte definir-lo $\vec{u} = \vec{u}(x, t)$, on x indica la posició i t el moment temporal. Així doncs, \vec{u} indica la velocitat de flux.

És important remarcar que gràcies a aquest camp vectorial de velocitat de flux, podem definir un **flux constant** de manera que el camp no varia amb el temps:

$$\frac{\partial \vec{u}}{\partial t} = 0 \quad (\text{A.11})$$

També podem distingir la compressibilitat del fluid amb la divergència del camp, ja que si aquesta és zero el flux serà **incompressible**.

$$\nabla \bullet \vec{u} = 0 \quad (\text{A.12})$$

I finalment també podem identificar si el fluid és o no irrotacional. Quan el rotacional del camp és zero, aquest és **irrotacional**.

$$\nabla \times \vec{u} = 0 \quad (\text{A.13})$$

És interessant destacar que $\nabla \times \vec{u}$ indica la tendència del fluid a causar vòrtexs; és a dir: la vorticitat, expressat amb la lletra ω .

A.2.2: L'equació

Tot seguit presento una de les formes de l'equació de Navier-Stokes. La qual analitzaré comparant-la amb la segona llei de Newton $\vec{F} = m\vec{a}$.

$$\rho \frac{D\vec{u}}{Dt} = \nabla \bullet \boldsymbol{\sigma} + \vec{f} \quad (\text{A.14})$$

El símil a fer és comprendre que ρ (notar lletra grega rho, no lletra p) identifica la densitat del fluid, i és l'equivalent de la massa en la segona llei de Newton, doncs denota la massa per unitat de volum; i $\frac{D\vec{u}}{Dt}$ es refereix a l'acceleració, doncs \vec{u} és la velocitat, i es coneix com la derivada material.

El terme $\nabla \bullet \boldsymbol{\sigma}$ identifica la divergència del tensor d'estres, o tensió. Aquesta mesura identifica la variació de forces aplicades a les diferents seccions (*cross-sections*), explicat a la secció [A.2.4](#).

Finalment, \vec{f} indica les forces externes aplicades al fluid, com la gravetat.

D'aquesta manera, podem concloure que les forces totals \vec{F} són equivalents a $\nabla \bullet \boldsymbol{\sigma} + \vec{f}$, el que es pot entendre com les forces externes més les forces elàstiques de retorn del material.

A.2.3: Derivada material

La derivada material descriu la ràtio de canvi del fluid, o d'un material, i depèn d'un camp de velocitats. Com que aquest ha de tenir en compte deformacions,

no ens serveix amb la derivada total. La derivada d'aquest terme s'analitzarà en profunditat a la secció A.5.2, però es defineix de la següent manera. Cal dir que les forces \vec{f} no són uniaxials, i a nivell de superfície.

$$\frac{D}{Dt} \equiv \frac{\partial}{\partial t} + \vec{u} \bullet \nabla \quad (\text{A.15})$$

D'aquesta manera, si es vol conèixer la ràtio de canvi del mateix camp de velocitats \vec{u} , obtenim el següent:

$$\frac{D\vec{u}}{Dt} = \frac{\partial \vec{u}}{\partial t} + \vec{u} \bullet \nabla \vec{u} \quad (\text{A.16})$$

El terme $\vec{u} \bullet \nabla \vec{u}$ es coneix com a **convecció**, equivalent al producte escalar de la velocitat pel gradient de la propietat. Aquest terme correspon a la variació d'un valor escalar per efecte d'un camp vectorial; és a dir: com varia el camp de velocitat en funció d'ell mateix en el nostre cas.

Aquesta component es pot exemplificar amb un tub que es fa estret, quan el líquid que circula per aquest convergeix en l'inici de la part estreta, la velocitat del fluid augmenta; i pel contrari, quan aquest es fa més ample, la velocitat del líquid disminueix per la **convecció**.

Aquest fet succeeix encara que el fluid es mogui a velocitat constant, és a dir: encara que $\frac{\partial \vec{u}}{\partial t} = 0$ és evident que es produeix una acceleració, per tant $\vec{u} \bullet \nabla \vec{u}$ produeix aquesta acceleració del fluid.

A.2.4: Tensió

Com s'ha mencionat abans, la tensió és la distribució de forces per unitat d'àrea tant en l'entorn d'un material, com en el seu interior. En casos molt simples la tensió és proporcional a l'estirament, com és el cas d'una molla amb les forces de retorn (Llei de Hooke).

Definim la tensió com a $\sigma = F/A$, tot i que usualment σ expressa concretament el tensor de tensió de Cauchy.

Tota tensió la podem descompondre en dues de diferents:

- Tensió normal: Aplicada perpendicularment a cadascuna de les superfícies.
- Tensió tallant: Aplicada tangencialment a cadascuna de les superfícies.

Per a poder descriure la segona, ens és necessari definir aquestes forces per cadascun dels plans. Això ho podem fer amb un tensor de rang 2.

$$\sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix} = [\vec{T}^{(e_1)} \quad \vec{T}^{(e_2)} \quad \vec{T}^{(e_3)}] \quad (\text{A.17})$$

Les components d'aquesta matriu es troben il·lustrades a la figura A.2.

La tensió va fortament lligada a la compressibilitat del fluid o material, perquè llavors s'ha de tenir en compte que el volum no és constant, i es compliquen les equacions considerablement. Per això, podem marcar la densitat com a ρ_0 en materials incompressibles, perquè ha passat a ser una constant en la derivació de l'equació de *Navier-Stokes*.

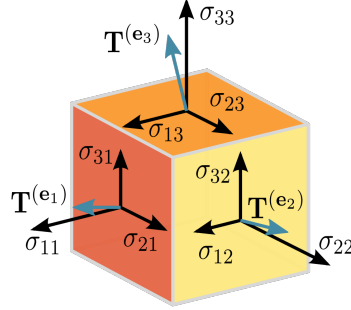


Figura A.2: *Components de la tensió en 3D*. Tensions en cadascun dels plans $T^{(e_i)}$, fletxa blava, amb les components corresponents, de manera que $T^{(e_i)} = (\sigma_{1i} \ \sigma_{2i} \ \sigma_{3i})^T$. (Imatge extreta de https://en.wikipedia.org/wiki/File:Components_stress_tensor_cartesian.svg, 8/12/2019)

La divergència del tensor de tensió, pot ser descomposta d'una manera diferent donada aquesta incompressibilitat:

$$\nabla \bullet \boldsymbol{\sigma} = -\vec{\nabla} p + \nabla \bullet \vec{\tau} \quad (\text{A.18})$$

On $\vec{\tau}$ identifica la tensió tallant, i p la pressió (diferenciar de ρ que indica densitat) que coincideix amb la tensió normal.

Analitzant el gradient de la pressió $-\vec{\nabla} p$, ens mostra que existeixen forces que busquen accelerar el flux de les zones de més pressió cap a les de menys pressió.

La tensió tallant $\vec{\tau}$ és molt particular, ja que per a fluids incompressibles aporta una nova component: la viscositat dinàmica μ , especificada com la força per temps a efectuar per unitat de superfície.

$$\vec{\tau} = \mu(\vec{\nabla} \vec{u} + \vec{\nabla} \vec{u}^T) \quad (\text{A.19})$$

Aquesta viscositat dinàmica acostuma a dependre de components com la densitat ρ o la pressió p , i representa una resistència a la deformació.

La seva divergència és definida per:

$$\nabla \bullet \vec{\tau} = \mu \nabla^2 \vec{u} \quad (\text{A.20})$$

Podem extrapolar que la tensió tallant funcionarà com una força de fregament amb el mateix fluid, d'acord amb el paràmetre μ i l'acceleració del fluid. Es pot entendre també que el moviment d'una porció de fluid depèn del moviment del fluid adjacent.

D'aquesta manera, si substituïm els termes de les equacions A.18 i A.20, obtenim:

$$\nabla \bullet \boldsymbol{\sigma} = -\vec{\nabla} p + \mu \nabla^2 \vec{u} \quad (\text{A.21})$$

A.2.5: Expansió de l'equació

Donada l'equació de Navier-Stokes, podem expandir-la amb els detalls de les seccions anteriors. Iniciant per modificar l'equació A.14 aplicant la densitat constant.

$$\rho \frac{D\vec{u}}{Dt} = \nabla \bullet \boldsymbol{\sigma} + \vec{f} \quad (\text{A.22})$$

Podem expandir el terme $\frac{D\vec{u}}{Dt}$ i la divergència $\nabla \bullet \boldsymbol{\sigma}$ amb les equacions A.16 i A.21 respectivament.

$$\rho \left(\frac{\partial \vec{u}}{\partial t} + \vec{u} \bullet \nabla \vec{u} \right) = -\vec{\nabla} p + \mu \nabla^2 \vec{u} + \vec{f} \quad (\text{A.23})$$

Si dividim entre ρ i restem el terme $\vec{u} \bullet \nabla \vec{u}$ obtenim l'equació objectiu.

$$\frac{\partial \vec{u}}{\partial t} = -\vec{u} \bullet \nabla \vec{u} - \frac{1}{\rho} \vec{\nabla} p + \frac{\mu}{\rho} \nabla^2 \vec{u} + \vec{f} \quad (\text{A.24})$$

Notar no hem especificat les unitats del terme \vec{f} , aquest l'hem anat arrossegant per tenir en compte totes les forces externes, sense tenir-les definides durant aquest procés.

Finalment, per a tenir l'equació final cal comentar el terme $\frac{\mu}{\rho}$, doncs és l'anomenada viscositat cinemàtica, representada amb la lletra ν amb unitats $\frac{\text{longitud}^2}{\text{temps}}$.

$$\nu = \frac{\mu}{\rho} \quad (\text{A.25})$$

Aplicant aquest terme final, obtenim la forma tradicional de les equacions de Navier-Stokes:

$$\frac{\partial \vec{u}}{\partial t} = -\vec{u} \bullet \nabla \vec{u} - \frac{1}{\rho} \vec{\nabla} p + \nu \nabla^2 \vec{u} + \vec{f} \quad (\text{A.26})$$

De manera que tenim una equació per calcular les variacions del camp vectorial de velocitats en funció del temps.

A mode de resum, tenim que:

- $-\vec{u} \bullet \nabla \vec{u}$: Convecció de la velocitat.
- $-\frac{1}{\rho} \vec{\nabla} p$: Dissipació per pressió.
- $\nu \nabla^2 \vec{u}$: Difusió de forces per viscositat.
- \vec{f} : Resta de forces externes.

A.3. Descomposició polar

El teorema de la descomposició polar ens diu que donada una matriu quadrada, o operador lineal com és el nostre tensor \mathbf{F} , podem produir una factorització en dues matrius. Posem l'exemple d'una matriu quadrada \mathbf{A} :

$$\mathbf{A} = \mathbf{R} \cdot \mathbf{U} \quad (\text{A.27})$$

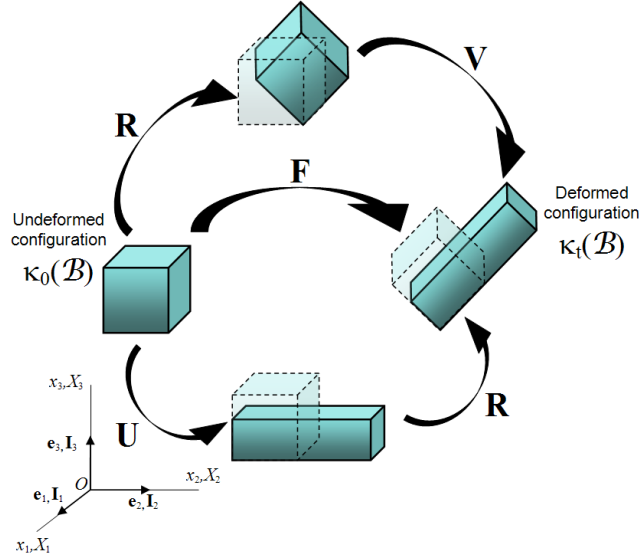


Figura A.3: *Descomposició polar del gradient de deformació*. Gradient \mathbf{F} descompost en una única matriu de rotació \mathbf{R} , i en els escalats propis de la descomposició polar. Podem veure com el cos \mathcal{B} és transformat d'un estat sense deformació κ_0 a un estat deformat κ_t després d'un temps t . (Imatge extreta de https://en.wikipedia.org/wiki/File:Polar_decomposition_of_F.png)

Descomposta en el producte d'una matriu \mathbf{R} unitària i una matriu \mathbf{U} positiva i hermítica (equivalent a la seva conjugada transposada).

En alt nivell podem dir que \mathbf{R} és una matriu de rotació, ja que al ser unitària no produeix cap escalat i a més té la propietat $\mathbf{R}^{-1} = \mathbf{R}^T$.

Per altra banda, \mathbf{U} és una matriu d'escalat sobre un conjunt d'eixos ortogonals.

És important notar que aquesta descomposició sempre existeix, però només és única quan \mathbf{A} és invertible. De manera que donada una matriu no invertible podem obtenir diferents descomposicions.

Una altra forma de descompondre la matriu \mathbf{A} és duent a terme la descomposició polar per l'esquerra:

$$\mathbf{A} = \mathbf{V} \cdot \mathbf{R} \quad (\text{A.28})$$

De manera que l'anterior equació A.27 és la descomposició polar per la dreta. En aquest cas el factor \mathbf{V} es pot computar a partir de les anteriors matrius:

$$\mathbf{V} = \mathbf{R} \cdot \mathbf{A} \cdot \mathbf{R}^{-1} = \mathbf{R} \cdot \mathbf{A} \cdot \mathbf{R}^T \quad (\text{A.29})$$

Si apliquem aquest fet al nostre gradient de deformació i el descomponem en $\mathbf{F} = \mathbf{R} \cdot \mathbf{U} = \mathbf{V} \cdot \mathbf{R}$ podem tractar el cos a deformar en diferents punts de la deformació, la figura A.3 il·lustra aquest fet.

A.4. Descomposició en valors singulars

De la mateixa manera que la descomposició polar, la descomposició en valors singulars és una factorització d'una matriu, no necessàriament quadrada, en 3 matrius diferents. Posem l'exemple de la descomposició de la matriu \mathbf{M} :

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \quad (\text{A.30})$$

En aquest cas, $\mathbf{\Sigma}$ és una matriu diagonal que realitza un escalat. Els valors de la diagonal d'aquest terme s'anomenen els valors singulars.

Per altra banda, \mathbf{U} i \mathbf{V}^* són matrius unitàries que representen una rotació sobre l'espai. En el cas concret de \mathbf{V}^* , representa la matriu transposada conjugada de \mathbf{V} , que en cas de ser una matriu quadrada unitària (com serà el cas amb \mathbf{F}) llavors $\mathbf{V}^{-1} = \mathbf{V}^*$. De manera que ens podem estalviar còmputos d'inverses:

$$\mathbf{U}\mathbf{U}^{-1} = \mathbf{U}\mathbf{U}^T = \mathbf{I} \quad (\text{A.31})$$

$$\mathbf{V}^*(\mathbf{V}^*)^{-1} = \mathbf{V}^*(\mathbf{V}^*)^T = \mathbf{I} \quad (\text{A.32})$$

Com que la descomposició en valors singulars factoritza una matriu en dues rotacions i un escalat, ens permeten també construir la descomposició polar, de manera que donada la següent descomposició polar per la drete de M :

$$\mathbf{M} = \mathbf{R} \cdot \mathbf{P} \quad (\text{A.33})$$

Es pot calcular la matriu de rotació unitària \mathbf{R} i d'escalat \mathbf{P} segons la descomposició de l'equació A.30 concatenant les rotacions per una banda, i fent l'escalat en l'espai de la rotació de \mathbf{V}^* :

$$\mathbf{R} = \mathbf{U}\mathbf{V}^* \quad (\text{A.34})$$

$$\mathbf{P} = \mathbf{V}\mathbf{\Sigma}\mathbf{V}^* \quad (\text{A.35})$$

Així doncs, veiem l'equivalència en l'ús de la descomposició en valors singulars i la descomposició polar.

Una propietat que ens aporta la descomposició en valors singulars, que la descomposició polar no ens aporta, és el fet de poder modificar únicament l'escalat d'una matriu, de manera que, per exemple, donada la descomposició del tensor de deformació \mathbf{F} , si reduïm o augmentem els valors singulars $\mathbf{\Sigma}$ estem afectant només a la magnitud de la deformació, no a la seva direcció.

A.5. Derivada del material

Per a dur a terme una simulació amb un material deformable, hem de saber derivar les diferents característiques d'aquest en funció del temps.

La més important per a una simulació és, potser, la velocitat no lineal de les deformacions.

A.5.1: Velocitat i Acceleració

Per això considerem l'equació del moviment com a:

$$\vec{x} = \phi(\vec{X}, t) \quad (2.1 \text{ revisitada})$$

De la qual podem extreure la velocitat de la partícula \vec{X} amb la derivada respecte el temps t

$$\mathcal{V}(\vec{X}, t) = \frac{\partial \phi(\vec{X}, t)}{\partial t} \quad (A.36)$$

On \mathcal{V} retorna un vector espacial, representant la velocitat.

Aquesta derivada de velocitat és en funció de la descripció *Lagrangiana* del material, però duent a terme la inversa de ϕ , seguint l'equació 2.2, obtenim la vessant *Euleriana*:

$$v(\vec{x}, t) = \mathcal{V}(\phi^{-1}(\vec{x}, t), t) \quad (A.37)$$

Notar la diferència de la funció velocitat entre \mathcal{V} i v , doncs la primera representa la versió *Lagrangiana* (una funció de \vec{X}) i la segona l'*Euleriana* (una funció de \vec{x}).

Aquesta tipologia de funcions també s'anomena el *push forward* (*Eulerià*) i *pull back* (*Lagrangià*), en aquest cas de la funció velocitat. Per exemple, la funció v de l'equació A.37 representa el *push forward* de la funció velocitat, i podem expressar el *pull back* de la mateixa com a:

$$\mathcal{V}(\vec{X}, t) = v(\phi(\vec{X}, t), t) \quad (A.38)$$

De la mateixa manera podríem representar l'acceleració:

$$\mathcal{A}(\vec{X}, t) = \frac{\partial^2 \phi(\vec{X}, t)}{\partial y^2} = \frac{\partial \mathcal{V}(\vec{X}, t)}{\partial t} \quad (A.39)$$

I el seu *push forward* (equació A.40) i *pull back* (equació A.41):

$$a(\vec{x}, t) = \mathcal{A}(\phi^{-1}(\vec{x}, t), t) \quad (A.40)$$

$$\mathcal{A}(\vec{X}, t) = a(\phi(\vec{X}, t), t) \quad (A.41)$$

A.5.2: Derivades materials

Donada una funció lagrangiana qualsevol \mathcal{G} definida segons les coordenades materials \vec{X} , concretament $\mathcal{G}(\vec{X}, t)$, podem denotar la seva derivada com a:

$$\frac{d\mathcal{G}}{dt} = \frac{\partial \mathcal{G}(\vec{X}, t)}{\partial t} \quad (A.42)$$

Aquesta, mesura el canvi associat a una partícula \vec{X} , però s'ha de tenir en compte els canvis que experimenta en funció de les seves coordenades espacials \vec{x} també, així que podem definir la derivada d'acord al seu *pull back* amb una funció alternativa g :

$$\frac{d\mathcal{G}(\vec{X}, t)}{dt} = \frac{\partial g(\phi(\vec{X}, t), t)}{\partial t} \quad (A.43)$$

Usant la regla de la cadena per a funcions multivariables, que podeu observar a continuació per a refrescar:

$$\frac{d}{dt}f(x(t), y(t)) = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt} \quad (\text{A.44})$$

Així que aplicant la regla de la cadena a la funció A.43 i usant l'equació 2.1:

$$\frac{d\mathcal{G}(\vec{\mathbf{X}}, t)}{dt} = \frac{\partial \mathcal{G}(\phi(\vec{\mathbf{X}}, t), t)}{\partial \vec{\mathbf{x}}} \frac{\partial \phi(\vec{\mathbf{X}}, t)}{\partial t} + \frac{\partial \mathcal{G}(\phi(\vec{\mathbf{X}}, t), t)}{\partial t} \quad (\text{A.45})$$

Un cop tenim una diferencial segons la descripció material, podem calcular-ne el *push forward* g per a poder computar la diferencial segons la descripció espacial. Notar el canvi de nomenclatura de derivada material en l'espai *Eulerià*.

$$\frac{Dg(\vec{\mathbf{x}}, t)}{Dt} = \frac{d\mathcal{G}(\phi^{-1}(\vec{\mathbf{x}}, t), t)}{dt} \quad (\text{A.46})$$

$$= \frac{\partial g(\phi(\phi^{-1}(\vec{\mathbf{x}}, t), t), t)}{\partial \vec{\mathbf{x}}} \frac{\partial \phi(\phi^{-1}(\vec{\mathbf{x}}, t), t)}{\partial t} + \frac{\partial g(\phi(\phi^{-1}(\vec{\mathbf{x}}, t), t), t)}{\partial t} \quad (\text{A.47})$$

Equació simplificable ja que per definició $\vec{\mathbf{x}} = \phi(\phi^{-1}(\vec{\mathbf{x}}, t), t)$,

$$\frac{Dg(\vec{\mathbf{x}}, t)}{Dt} = \frac{\partial g(\vec{\mathbf{x}}, t)}{\partial \vec{\mathbf{x}}} \frac{\partial \phi(\phi^{-1}(\vec{\mathbf{x}}, t), t)}{\partial t} + \frac{\partial g(\vec{\mathbf{x}}, t)}{\partial t} \quad (\text{A.48})$$

Gràcies a les equacions A.36 i A.37 podem introduir la velocitat com a terme de la derivada:

$$\frac{Dg(\vec{\mathbf{x}}, t)}{Dt} = \frac{\partial g(\vec{\mathbf{x}}, t)}{\partial \vec{\mathbf{x}}} v(\vec{\mathbf{x}}, t) + \frac{\partial g(\vec{\mathbf{x}}, t)}{\partial t} \quad (\text{A.49})$$

És curiós veure com, qualsevol mena de derivada serà influenciada per la velocitat del cos. Finalment podem substituir la derivada de la posició pel mateix gradient i reorganitzar:

$$\frac{Dg(\vec{\mathbf{x}}, t)}{Dt} = v(\vec{\mathbf{x}}, t) \nabla g + \frac{\partial g(\vec{\mathbf{x}}, t)}{\partial t} \quad (\text{A.50})$$

Aquesta equació és exactament la mateixa que el terme de convecció de l'equació de Navier-Stokes vista en la secció A.2.3, si substituïm g per la velocitat (o pel seu camp vectorial).

Fixeu-vos que podem usar aquesta nova derivada per a trobar la funció acceleració a amb la derivada material, perquè:

$$a = \frac{Dv}{Dt} \quad (\text{A.51})$$

I substituint termes a l'equació A.49, trobem un resultat diferent a l'esperat:

$$a(\vec{\mathbf{x}}, t) = \frac{\partial v(\vec{\mathbf{x}}, t)}{\partial \vec{\mathbf{x}}} v(\vec{\mathbf{x}}, t) + \frac{\partial v(\vec{\mathbf{x}}, t)}{\partial t} \quad (\text{A.52})$$

Ja que és senzill veure com $a(\vec{\mathbf{x}}, t) \neq \frac{\partial v(\vec{\mathbf{x}}, t)}{\partial t}$, fet que no resulta gens intuïtiu, però es pot entendre com què l'acceleració produïda a una porció de l'espai depèn de l'espai del voltant.

A.6. Esforç (Strain)

L'esforç (o strain en anglès) identifica l'elongació o compressió relativa en un material o cos, i s'expressa amb la lletra grega ϵ . L'exemple més senzill per visualitzar aquest concepte és una molla, on el seu esforç és identificat en funció de la seva elongació inicial L_0 i la deformada L :

$$\epsilon = \frac{\Delta L_0}{L_0} = \frac{L - L_0}{L_0} \quad (\text{A.53})$$

El problema rau en què l'exemple de la molla és unidimensional, i el problema que volem tractar és tridimensional. En aquests casos l'esforç no té per què ser lineal, ni estar distribuït uniformement en el cos; de manera que no podem expressar l'esforç com un escalar, però sí com un tensor.

$$\boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_{xx} & \epsilon_{xy} & \epsilon_{xz} \\ \epsilon_{yx} & \epsilon_{yy} & \epsilon_{yz} \\ \epsilon_{zx} & \epsilon_{zy} & \epsilon_{zz} \end{bmatrix} \quad (\text{A.54})$$

Com que l'esforç depèn de la posició en l'espai, podem també definir-lo com una aplicació $\boldsymbol{\epsilon} = \boldsymbol{\epsilon}(\vec{x})$, en funció d'una coordenada \vec{x} , la qual és només equivalent a zero si no hi ha hagut cap deformació.

A.6.1: Caracterització de l'esforç

Podem concloure que l'esforç depèn del desplaçament produït entre el cos deformat i sense deformat. En el nostre cas usarem el camp vectorial de desplaçament \vec{u} , que podem caracteritzar *Lagrangianament* com una funció de la següent manera:

$$\vec{u}(\vec{\mathbf{X}}, t) = \mathbf{b}(t) + \phi(\mathbf{X}, t) - \vec{\mathbf{X}} \quad (\text{A.55})$$

On $\vec{b}(t)$ és el desplaçament produït a tot el cos de manera generalitzada, com d'un sòlid rígid.

Però realment no ens interessa el desplaçament produït, sinó com canvia aquest respecte al mateix material. És a dir, el gradient respecte de la posició, que podem obtenir amb la derivada següent.

$$\frac{d\vec{u}}{d\vec{\mathbf{X}}} = \frac{d\phi}{d\vec{\mathbf{X}}} - \frac{\partial \vec{\mathbf{X}}}{\partial \vec{\mathbf{X}}} = \mathbf{F} - \mathbf{I} \quad (\text{A.56})$$

On \mathbf{F} és el gradient de deformació obtingut de l'equació 2.4.

També podem realitzar les mateixes operacions respecte de l'espai *Eulerià*, duent a terme el *push forward* de \vec{u} :

$$\vec{U}(\vec{\mathbf{x}}, t) = \mathbf{b}(t) + \phi - \phi^{-1}(\vec{\mathbf{x}}, t) \quad (\text{A.57})$$

$$\frac{D\vec{U}}{D\vec{\mathbf{x}}} = \frac{\partial \vec{\mathbf{x}}}{\partial \vec{\mathbf{x}}} - \frac{D\phi^{-1}}{D\vec{\mathbf{x}}} = \mathbf{I} - \mathbf{F}^{-1} \quad (\text{A.58})$$

És senzill veure com ambdós espais busquen la diferència entre el gradient de deformació i la identitat, que en altres paraules, si no hi ha hagut deformació, aquesta diferència és zero, però si hi ha hagut deformació, aquesta marca la tendència del desplaçament.

A.6.2: Tensors d'esforç finits

Dur a terme el càlcul de l'esforç produït localment no és trivial, i més quan es volen tractar també deformacions importants. Una de les maneres de solucionar aquest problema és derivar el tensor d'esforç de *Green-Lagrange*, equació A.67 en aquesta mateixa secció.

Una mesura típica de deformació és la diferència entre les posicions diferencials tant en el cos deformat com en el sense deformar $D\vec{x} - d\vec{X}$, però com que l'esforç no sempre és lineal és molt més acurat usar les diferencials al quadrat $D\vec{x}^2 - d\vec{X}^2$, on aquest quadrat $D\vec{x}^2$ és el producte escalar amb si mateix. De manera que definim l'esforç no lineal com a:

$$\epsilon = \frac{D\vec{x}^2 - d\vec{X}^2}{2d\vec{X}^2} \quad (\text{A.59})$$

El terme $\frac{1}{2}$ s'explica en l'equació A.75, de manera intuïtiva.

Aquesta equació segueix sent difícil de computar, ja que encara necessitem valors infinitesimals. Així que desgranem el terme *Eulerià*, amb l'objectiu d'eliminar-lo, i per això recordem que l'equació 2.6 permet extreure el terme $D\vec{x}$:

$$D\vec{x}^2 = D\vec{x}^T \cdot D\vec{x} \quad (\text{A.60})$$

$$= (\mathbf{F} \cdot d\vec{X})^T \cdot \mathbf{F} \cdot d\vec{X} \quad (\text{A.61})$$

$$= d\vec{X}^T \cdot \mathbf{F}^T \mathbf{F} \cdot d\vec{X} \quad (\text{A.62})$$

$$= d\vec{X}^T \cdot \mathbf{C} \cdot d\vec{X} \quad (\text{A.63})$$

També hem substituït el producte dels gradients de deformació pel tensor dret de deformació de *Cauchy-Green* de l'equació 2.13.

D'aquesta manera podem desenvolupar el tensor de l'equació A.59 amb els resultats de l'equació A.63:

$$D\vec{x}^2 - d\vec{X}^2 = d\vec{X}^T \cdot \mathbf{C} \cdot d\vec{X} - d\vec{X}^T \cdot \vec{X} \quad (\text{A.64})$$

$$= d\vec{X}^T \cdot (\mathbf{C} - \mathbf{I}) \cdot d\vec{X} \quad (\text{A.65})$$

$$= d\vec{X}^T \cdot 2\epsilon \cdot d\vec{X} \quad (\text{A.66})$$

On hem realitzat la substitució $2\epsilon = \mathbf{C} - \mathbf{I}$. D'aquesta manera podem tornar a arranjar l'equació i obtenir el tensor d'esforç de *Green-Lagrange*:

$$\epsilon = \frac{1}{2}(\mathbf{C} - \mathbf{I}) = \frac{D\vec{x}^2 - d\vec{X}^2}{2d\vec{X}^2} \quad (\text{A.67})$$

Aquesta equació la podem desgranar, expandint el terme \mathbf{C} tensor de *Cauchy-Green*, equació 2.13, i les expansions del camp de desplaçament de l'equació A.56:

$$\boldsymbol{\epsilon} = \frac{1}{2}(\mathbf{C} - \mathbf{I}) \quad (\text{A.68})$$

$$= \frac{1}{2}(\mathbf{F}^T \mathbf{F} - \mathbf{I}) \quad (\text{A.69})$$

$$= \frac{1}{2}(((\frac{d\vec{u}}{d\vec{X}})^T + \mathbf{I})(\frac{d\vec{u}}{d\vec{X}} + \mathbf{I}) - \mathbf{I}) \quad (\text{A.70})$$

$$= \frac{1}{2}((\frac{d\vec{u}}{d\vec{X}})^T + \frac{d\vec{u}}{d\vec{X}} + (\frac{d\vec{u}}{d\vec{X}})^T \cdot \frac{d\vec{u}}{d\vec{X}}) \quad (\text{A.71})$$

D'aquesta manera obtenim el tensor no lineal d'esforç de *Green* ϵ_G , i eliminant el terme quadràtic obtenim la seva linearització ϵ_C , el tensor lineal d'esforç de *Cauchy*:

$$\epsilon_G = \frac{1}{2}((\frac{d\vec{u}}{d\vec{X}})^T + \frac{d\vec{u}}{d\vec{X}} + (\frac{d\mathbf{u}}{d\vec{X}})^T \cdot \frac{d\vec{u}}{d\vec{X}}) \quad (\text{A.72})$$

$$\epsilon_C = \frac{1}{2}((\frac{d\vec{u}}{d\vec{X}})^T + \frac{d\vec{u}}{d\vec{X}}) \quad (\text{A.73})$$

Per explicar el terme de $\frac{1}{2}$ aplicat als tensors anteriors, podem mirar com queda la matriu respectiva. Per això veiem ϵ_C , que és molt més senzill, on cada element de la matriu $\epsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i})$, on $u_{i,j}$ significa el gradient de u sobre la coordenada i -èssima, entre la parcial de la coordenada j .

$$\epsilon_C = \begin{bmatrix} \frac{\partial u_x}{\partial X} & \frac{1}{2}(\frac{\partial u_x}{\partial Y} + \frac{\partial u_y}{\partial X}) & \frac{1}{2}(\frac{\partial u_x}{\partial Z} + \frac{\partial u_z}{\partial X}) \\ \frac{1}{2}(\frac{\partial u_y}{\partial X} + \frac{\partial u_x}{\partial Y}) & \frac{\partial u_y}{\partial Y} & \frac{1}{2}(\frac{\partial u_y}{\partial Z} + \frac{\partial u_z}{\partial Y}) \\ \frac{1}{2}(\frac{\partial u_z}{\partial X} + \frac{\partial u_x}{\partial Z}) & \frac{1}{2}(\frac{\partial u_z}{\partial Y} + \frac{\partial u_y}{\partial Z}) & \frac{\partial u_z}{\partial Z} \end{bmatrix} \quad (\text{A.74})$$

Podem veure com el terme $\frac{1}{2}$ serveix per compensar el fet de sumar dues vegades les parcials, de manera que evitem mesurar el doble de la desviació produïda respecte al material original, del qual estem mesurant l'esforç.

Això mateix es pot realitzar per ϵ_G , tot i que és severament més complicat de veure.

Una altra manera de veure-ho és que si no apliquem el terme $\frac{1}{2}$ a l'equació A.59, per deformacions molt petites on $D\vec{x} \approx d\vec{X}$:

$$\epsilon = \frac{D\vec{x}^2 - d\vec{X}^2}{d\vec{X}^2} = \frac{D\vec{x} - d\vec{X}}{d\vec{X}} \frac{D\vec{x} + d\vec{X}}{d\vec{X}} \approx 2 \frac{D\vec{x} - d\vec{X}}{d\vec{X}} \quad (\text{A.75})$$

Aquests tensors d'esforç són *Lagrangians*, però també se'n podrien extreure els respectius *Eulerians* usant coordenades espacials com a referència, duent a terme el mateix procediment efectuat amb els primers.

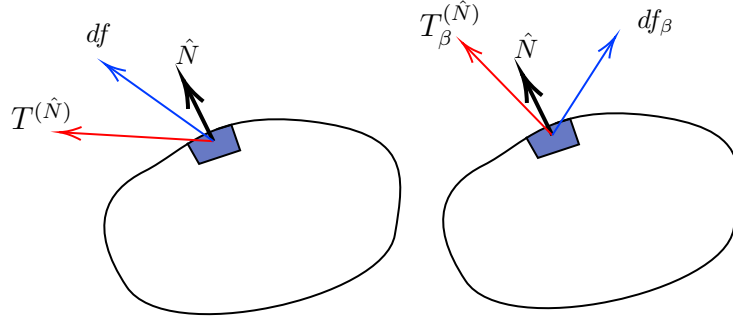


Figura A.4: *Tracció sobre el mateix cos*. Mateix cos de referència, en què se li apliquen dues forces diferents df i df_β sobre una mateixa superfície (identificada en blau), resultant en una diferent tracció superficial. (Elaboració pròpia)

A.7. Segon tensor de *Piola-Kirchhoff*

Per entendre aquest tensor, suposem que tenim un cos amb una porció de superfície dS (amb normal \hat{N}), en la que en dos instants de temps diferents se li apliquen distintes forces $d\vec{f}$ i $d\vec{f}_\beta$, la qual cosa provoca dues traccions diferents $\vec{T}^{(\hat{N})}$ i $\vec{T}_\beta^{(\hat{N})}$, respectivament, observables a la figure A.4.

La idea per relacionar les dues forces $d\vec{f}$ i $d\vec{f}_\beta$, és que existeix un gradient de deformació \mathbf{F} que altera aquesta força $d\vec{f}_\beta$ per a “deformar-la” fins a $d\vec{f}$:

$$d\vec{f} = \mathbf{F} \cdot d\vec{f}_\beta \quad (\text{A.76})$$

A més, sabem per l’equació 2.31 què $d\vec{f}_\beta = \vec{T}_\beta^{(\hat{N})} \cdot dS$, i que podem calcular $\vec{T}_\beta^{(\hat{N})}$ a partir d’algun tensor \mathbf{S} multiplicat per la normal de la superfície, igual que el mostrat a l’equació 2.30 i 2.32.

$$\vec{T}_\beta^{(\hat{N})} = \mathbf{S} \cdot \hat{N} \quad (\text{A.77})$$

D’aquesta manera, substituint a l’equació A.76, obtenim la força en funció de la deformació i la superfície del material:

$$d\vec{f} = \mathbf{F} \cdot \mathbf{S} \cdot \hat{N} \cdot dS \quad (\text{A.78})$$

$$d\vec{f} = \mathbf{F} \cdot \mathbf{S} \cdot \vec{N} \quad (\text{A.79})$$

Equivalentment amb el primer tensor de tensió de *Piola-Kirchhoff*, podem dir que les forces $d\vec{f}$ són equivalents, de manera que podem igualar amb l’equació 2.46, i desenvolupar:

$$\mathbf{P} \cdot d\vec{S} = \mathbf{F} \cdot \mathbf{S} \cdot \vec{N} \quad (\text{A.80})$$

$$\mathbf{P} = \mathbf{F} \cdot \mathbf{S} \quad (\text{A.81})$$

$$\mathbf{S} = \mathbf{F}^{-1} \cdot \mathbf{P} \quad (\text{A.82})$$

Notar la relació entre els dos tensors de tensió de l'equació A.82.

Finalment podem substituir el tensor \mathbf{P} per l'equació 2.49, i obtenir el segon tensor de *Piola-Kirchhoff* \mathbf{S} :

$$\mathbf{S} = J \cdot \mathbf{F}^{-1} \cdot \boldsymbol{\sigma} \cdot \mathbf{F}^{-T} \quad (\text{A.83})$$

Aquest tensor, a diferència del primer tensor de *Piola-Kirchhoff*, sí que és simètric doncs es pot comprovar que $\mathbf{S} = \mathbf{S}^T$.

Amb aquest nou tensor podrem computar únicament amb descripció *Lagrangiana* les forces producte d'una deformació, sense haver de computar la deformació en si.

A.8. Equacions de govern

Donat un model matemàtic, les equacions de govern descriuen el comportament de certes variables dependents del nostre sistema.

Aquestes s'utilitzen per mantenir l'estabilitat del model i per garantir que funciona d'acord amb uns principis determinats, com poden la ser conservació del moment angular, o de l'energia.

En el cas de simulacions de fluids, ens interessa la conservació de la massa i del moment lineal.

A.8.1: Conservació de la massa

Els fluids són constants en el que a massa es refereix. Aquesta no es destrueix, encara que pot variar en densitat.

Per això voldríem alguna equació per a simbolitzar aquest fet de manera senzilla, a partir de la variació de densitat.

Donades les densitats R i ρ , les podem representar com els respectius *pull back* i *push forward*, en l'instant de temps t :

$$R(\vec{\mathbf{X}}, t) = \rho(\phi(\vec{\mathbf{X}}, t), t) \quad (\text{A.84})$$

$$\rho(\vec{\mathbf{x}}, t) = R(\phi^{-1}(\vec{\mathbf{x}}, t), t) \quad (\text{A.85})$$

Per poder tractar la densitat, necessitem un volum. Per això usarem una bola B_ϵ^t de radi ϵ al voltant d'un punt arbitrari de l'espai en l'instant t temps.

De manera que podem definir la densitat en funció d'aquesta bola.

$$\rho(B_\epsilon^t, t) = \frac{\text{massa}(B_\epsilon^t)}{\text{volum}(B_\epsilon^t)} \quad (\text{A.86})$$

La qual podem expressar també a escala infinitesimal, si diem que la bola B_ϵ^t està centrada en un punt *Eulerià* $\vec{\mathbf{x}}$, amb el radi de la bola tendint a zero.

$$\rho(\vec{\mathbf{x}}, t) = \lim_{\epsilon \rightarrow +0} \frac{\text{massa}(B_\epsilon^t)}{\text{volum}(B_\epsilon^t)} = \lim_{\epsilon \rightarrow +0} \frac{\text{massa}(B_\epsilon^t)}{\int_{B_\epsilon^t} d\vec{\mathbf{x}}} \quad (\text{A.87})$$

On $d\vec{x}$ representen porcions infinitesimals del volum de B_ϵ^t .

D'aquesta manera podem aïllar la massa de la bola, i computar la variació de massa des de l'instant t fins l'instant 0. Per brevetat, ometré el límit $\epsilon \rightarrow +0$.

$$\text{mass}(B_\epsilon^t) = \int_{B_\epsilon^t} \rho(\vec{x}, t) d\vec{x} \quad (\text{A.88})$$

Podem transformar de descripció *Euleriana* a *Lagrangiana* gràcies a l'equació 2.5 en el moment de temps 0, relacionat amb el determinant J .

$$\text{mass}(B_\epsilon^t) = \int_{B_\epsilon^t} \rho(\vec{x}, t) d\vec{x} \quad (\text{A.88})$$

$$= \int_{B_\epsilon^0} J \cdot R(\vec{X}, t) d\vec{X} \quad (\text{A.89})$$

$$= \int_{B_\epsilon^0} R(\vec{X}, 0) d\vec{X} \quad (\text{A.90})$$

$$= \text{mass}(B_\epsilon^0) \quad (\text{A.91})$$

El que es tradueix en què la massa en una bola arbitrària no ha de canviar amb el temps, tot i la variació de densitat o d'espai que ocupa el material.

Atès que la bola és arbitrària, s'ha de complir per tot l'espai. Per tant, per tot instant de temps $t \geq 0$:

$$J(\vec{X}, t) \cdot R(\vec{X}, t) = R(\vec{X}, 0) \quad (\text{A.92})$$

Que es pot escriure també com a:

$$\frac{\partial(J(\vec{X}, t) \cdot R(\vec{X}, t))}{\partial t} = 0 \quad (\text{A.93})$$

Per altra banda, expandint i diferenciant, podem obtenir el seu vessant *Eulerià* [7]:

$$\frac{D\rho(\vec{x}, t)}{Dt} + \rho(\vec{x}, t) \nabla \bullet \vec{v}(\vec{x}, t) = 0 \quad (\text{A.94})$$

A.8.2: Conservació del moment lineal

El moment lineal, expressat com a velocitat per unitat de massa, ha de ser constant en tot moment, només sent afectat per forces externes i internes. No es pot dissipar.

Aquest fet el podem expressar com l'equació de *Navier-Stokes*, consultable a A.2.2, aplicada a la velocitat:

$$\rho(\vec{x}, t) \frac{D\vec{v}}{Dt}(\vec{x}, t) = \nabla \bullet \boldsymbol{\sigma}(\vec{x}, t) + \vec{f}(\vec{x}, t) \quad (\text{A.95})$$

Normalment, només tindrem en compte la gravetat \vec{g} com a força externa, per tant:

$$\rho(\vec{x}, t) \frac{D\vec{v}}{Dt}(\vec{x}, t) = \nabla \bullet \boldsymbol{\sigma}(\vec{x}, t) + \rho \vec{g} \quad (\text{A.96})$$

A.8.3: Formulació dèbil

Una formulació dèbil d'una funció o equació diferencial, és una aplicació alternativa que permet l'anàlisi des de l'àlgebra lineal en un espai vectorial. Aquesta formulació es fa respecte a una funció de "test" arbitrària, funcions substitutives de derivades que no existeixen o no són calculables, que serveix com a punt de referència de la nova funció.

Com que aquesta formulació és complexa, i se surt de l'abast d'aquest document, refereixo al curs de Jiang et al. [7] o alternativament al de Bonet [14]. De manera il·lustrativa, presento l'equació dèbil que s'utilitzarà per resoldre el sistema, i simular el fluid en qüestió, donada una funció arbitrària \vec{Q} *Lagrangiana* o \vec{q} *Euleriana*.

$$\int_{\Omega^0} \vec{Q} \cdot R \cdot \vec{A} \cdot d\vec{X} = \int_{\partial\Omega^t} \vec{q} \cdot \vec{t} \cdot ds(\vec{x}) - \int_{\Omega^t} \vec{q} \cdot \boldsymbol{\sigma} \cdot d\vec{x} \quad (\text{A.97})$$

On la part esquerra de l'equació identifica les components de densitat R i acceleració \vec{A} , en descripció *Lagrangiana* i en el moment de temps inicial 0, sense deformar. La integral respecte Ω^0 identifica tot el cos o material Ω en l'instant inicial.

Per altra banda, a l'esquerra de l'equació trobem una primera integral sobre la superfície del material deformat Ω^t (marquem la superfície amb el símbol ∂), sobre una tracció \vec{t} i una diferencial de la superfície en un punt $ds(\vec{x})$ *Eulerià*. Finalment es compensa amb la diferència de les forces, on $\boldsymbol{\sigma}$ és la tensió de *Cauchy*.

En gràfics per computador no és necessari una simulació acurada d'aquesta equació, i amb una equació similar n'és normalment suficient.

B. Simulacions disponibles

En aquest apèndix es descriuen tots els arxius *GIF* i *SBF* (secció 6.2) que s’han creat per a il·lustrar les capacitats del simulador de fluids *MPM-MLS* implementat en aquest treball.

Totes les animacions es poden trobar en el la següent carpeta de [Google Drive](https://drive.google.com/drive/folders/1t-WDatICCti9cPGHokXEi5B0oFrqEd-R) (També accessible amb l’enllaç <https://drive.google.com/drive/folders/1t-WDatICCti9cPGHokXEi5B0oFrqEd-R> [Gener 2020]).

Per cada *GIF* s’ofereix una petita explicació i detalls de la simulació darrere el mateix.

Cal dir que algunes simulacions son antigues, i estan gravades amb les partícules com a cubs, en comptes d’icosaedres.

Tots aquells paràmetres mancants a les taules, s’assumeixen els bàsics següents:

E_0	10^5
ν_0	0.3
hardening ₀	10
vol ₀	1
mass ₀	1
plasticitat ₀	activada
$t_{c,0}$	$2.5 \cdot 10^{-2}$
$t_{s,0}$	$7.5 \cdot 10^{-3}$

Finalment, totes les simulacions han estat dutes a terme amb una graella de $128 \times 128 \times 128$

B.1: 2d

Disponible a [enllaç](#).

Simulació bidimensional de prova. Els colors es basen únicament en l’alçada inicial de la partícula. La pressió a les cantonades provoca l’augment de flux en direcció oposada.

E	$3 \cdot 10^4$
ν	0.4
hardening	10
Δt	10^{-3}

B.2: explode

Disponible a [enllaç](#).

Simulació que “explota” degut a un Δt massa elevat.

E	$3 \cdot 10^6$
ν	0.3
Δt	$3 \cdot 10^{-5}$
model	corrotacional

B.3: noQ

Disponible a [enllaç](#).

Col·lisió entre dos sòlids sense cap model d'energia elàstica. Al no existir cap tipus d'energia entre partícules, aquestes es deixen portar per la mateixa velocitat de l'entorn.

E	10^3
ν	0.35
hardening	8
Δt	10^{-5}
partícules	80776
model	cap

B.4: noQY

Disponible a [enllaç](#).

Mateixa simulació que l'anterior B.3, però amb model d'energia elàstica corrotacional fix.

E	10^3
ν	0.35
hardening	8
Δt	10^{-5}
partícules	80776
model	corrotacional

B.5: cmp1

Disponible a [enllaç](#).

Tres cubs amb diferents mòduls de Young, que provoquen diferents reaccions a l'impacte amb el terra. Valors d'esquerra a dreta.

E_1	300
E_2	800
E_3	1300
ν	0.35
hardening	10
Δt	10^{-5}
partícules	98304
model	corrotacional

B.6: cmp2

Disponible a [enllaç](#).

Tres cubs amb diferents mòduls de Young, provocant rebots a la superfície.

E_1	800
E_2	5000
E_3	10^4
ν	0.3
hardening	10
Δt	10^{-5}
partícules	98304
model	corrotacional

B.7: cmp4

Disponible a [enllaç](#).

Tres cubs amb diferents coeficients ν , de manera que la distensió provoca trencament del material.

E	10^3
ν_1	0.1
ν_2	0.35
ν_3	0.45
hardening	10
Δt	10^{-5}
partícules	98304
model	corrotacional

B.8: cor2

Disponible a [enllaç](#).

Tres cubs del mateix material, llançats des de diferents alçades, que interactuen parcialment els uns amb els altres (acoblament), modificant la seva deformació plàstica.

E	$1.5 \cdot 10^3$
ν_1	0.3
Δt	10^{-5}
partícules	59049
model	corrotacional

B.9: prog2

Disponible a [enllaç](#).

Tres cubs de materials molt tous, situats en diferents profunditats i alçades, amb velocitats inicials, impacten contra una paret. Ruptura progressiva del material.

E	10^3
ν	0.3
Δt	10^{-5}
partícules	98304
model	corrotacional

B.10: prog3

Disponible a [enllaç](#).

Tres cubs, situats en diferents profunditats i alçades, amb velocitats inicials elevades, impacten contra una paret, i els uns amb els altres, provocant el trencament de la superfície i petites deformacions plàstiques.

E	10^4
ν	0.3
Δt	10^{-5}
partícules	98304
model	corrotacional

B.11: prog4

Disponible a [enllaç](#).

Tres cubs rígids, situats en diferents profunditats i alçades, amb velocitats inicials, impacten contra una paret. Deformació per colpejament. Notar el alt rebot del cub magenta.

E	$4 \cdot 10^4$
ν	0.3
Δt	10^{-5}
partícules	98304
model	corrotacional

B.12: prog5

Disponible a [enllaç](#).

Tres cubs rígids, situats en diferents profunditats i alçades, amb velocitats inicials, impacten contra una paret. Rebot elevat entre cubs.

E	10^5
ν	0.3
Δt	10^{-5}
partícules	98304
model	corrotacional

B.13: 3fast

Disponible a [enllaç](#).

Tres cubs idèntics i rígids, llançats fortament contra una paret des de diferents distàncies.

E	10^5
ν	0.3
Δt	10^{-5}
partícules	46875
model	corrotacional

B.14: nPlastic0

Disponible a [enllaç](#).

Impacte de dues esferes totalment elàstiques.

E	10^3
ν	0.35
plasticitat	desactivada
Δt	10^{-5}
partícules	80776
model	corrotacional

B.15: nPlastic6

Disponible a [enllaç](#).

Impacte de dues esferes totalment elàstiques, amb gran deformació axial.

E	10^2
ν	0.45
plasticitat	desactivada
Δt	10^{-5}
partícules	80776
model	corrotacional

B.16: nPlastic7

Disponible a [enllaç](#).

Impacte de dues esferes totalment elàstiques, amb gran deformació axial. Idèntic a l'anterior, però *Neo-Hookean*.

E	10^2
ν	0.45
plasticitat	desactivada
Δt	10^{-5}
partícules	80776
model	<i>Neo-Hookean</i>

B.17: sand2

Disponible a [enllaç](#).

Material dens i tou, amb interaccions físiques (model físic *two slopes unbounded*). Veure el reposicionament del mateix, d'acord amb les col·lisions.

E	400
ν	0.3
Δt	10^{-5}
partícules	104280
model	corrotacional

B.18: sand3

Disponible a [enllaç](#), juntament amb una versió amb diferent perspectiva a [enllaç](#).

Material trencable i deformable, amb interaccions físiques (model físic *two slopes unbounded*). Veure el trencament per trossos, i la fragmentació del mateix.

E	400
ν	0.3
Δt	10^{-5}
partícules	104280
model	corrotacional

B.19: sand4

Disponible a [enllaç](#).

Material deformable, però prou rígid per a que es trenqui, amb interaccions físiques (model físic *two slopes unbounded*). Trencament del material en tres trossos evidents, i fractures dels laterals.

E	800
ν	0.3
Δt	10^{-5}
partícules	104280
model	corrotacional

B.20: hoursand1

Disponible a [enllaç](#).

Material requerint poca energia per a ser deformat i senzill de trencar, amb interaccions físiques (model físic *two slopes*).

E	10
ν	0.3
Δt	10^{-5}
partícules	104280
model	corrotacional

B.21: hoursand2

Disponible a [enllaç](#).

Material requerint poca energia per a ser deformat, però amb plasticitat que activa l'enduriment del material degut a la compressió d'aquest (model físic *two slopes*).

E	100
ν	0.3
Δt	10^{-5}
partícules	104280
model	corrotacional

B.22: hoursand3

Disponible a [enllaç](#).

Material a semi rígid, que per la compressió d'aquest i el seu coeficient E , és trenca en trossos evidents (model físic *two slopes*).

E	400
ν	0.3
Δt	10^{-5}
partícules	104280
model	corrotacional

B.23: snow1

Disponible a [enllaç](#).

Impacte de dues figures rígides, rebot no plàstic i sense deformació.

E	$1.4 \cdot 10^4$
ν	0.2
Δt	10^{-5}
partícules	57536
model	corrotacional

B.24: snow2

Disponible a [enllaç](#).

Impacte de dues figures molt deformables, de manera que actuen com un fluid al trencar-se senzillament.

E	10^3
ν	0.39
hardening	10
Δt	10^{-5}
partícules	57536
model	corrotacional

B.25: snow2.2

Disponible a [enllaç](#).

Impacte de dues figures, amb diferents moduls de Young, on un d'ells s'acobla al segon.

E_1	10^4
E_2	$2 \cdot 10^3$
ν	0.3
hardening	10
Δt	10^{-5}
partícules	57536
model	corrotacional

B.26: snow2.3

Disponible a [enllaç](#).

Impacte de dues figures, amb diferents moduls de Young. Bona vista del trencament de l'objecte.

E_1	10^4
E_2	$3 \cdot 10^3$
ν	0.3
hardening	10
Δt	10^{-5}
partícules	57536
model	corrotacional

B.27: snow2.7

Disponible a [enllaç](#).

Impacte de dues figures, amb diferents moduls de Young. Liquació amb trencament. Bona expansió com a fluid.

E_1	10^4
E_2	$3 \cdot 10^3$
ν_1	0.3
ν_2	0.36
hardening	10
Δt	10^{-5}
partícules	57536
model	corrotacional

B.28: snow2.8

Disponible a [enllaç](#).

Impacte d'objecte "fluid dens", amb partícules amb el doble de massa, contra objecte sòlid. Veure l'arrossegament del pes de l'objecte.

E_1	10^4
E_2	$3 \cdot 10^3$
ν_1	0.3
ν_2	0.36
mass ₁	1
mass ₂	2
hardening	10
Δt	10^{-5}
partícules	57536
model	corrotacional

B.29: snow3.2

Disponible a [enllaç](#).

Impacte de dues figures, amb diferents moduls de Young, on un d'ells s'acobla al segon. *Neo-Hookean*.

E_1	10^4
E_2	$2 \cdot 10^3$
ν	0.3
hardening	10
Δt	10^{-5}
partícules	57536
model	<i>Neo-Hookean</i>

B.30: snow3.3

Disponible a [enllaç](#).

Impacte de dues figures, amb diferents moduls de Young. Bona vista del trencament de l'objecte. *Neo-Hookean*.

E_1	10^4
E_2	$3 \cdot 10^3$
ν	0.3
hardening	10
Δt	10^{-5}
partícules	57536
model	<i>Neo-Hookean</i>

B.31: snow3.7

Disponible a [enllaç](#).

Impacte de dues figures, amb diferents moduls de Young. Lliquació amb trencament. Bona expansió com a fluid. *Neo-Hookean*.

E_1	10^4
E_2	$3 \cdot 10^3$
ν_1	0.3
ν_2	0.36
hardening	10
Δt	10^{-5}
partícules	57536
model	<i>Neo-Hookean</i>

B.32: snow3.8

Disponible a [enllaç](#).

Impacte d'objecte "fluid dens", amb partícules amb el doble de massa, contra objecte sòlid. Veure l'arrossegament del pes de l'objecte. *Neo-Hookean*.

E_1	10^4
E_2	$3 \cdot 10^3$
ν_1	0.3
ν_2	0.36
mass ₁	1
mass ₂	2
hardening	10
Δt	10^{-5}
partícules	57536
model	<i>Neo-Hookean</i>

B.33: hard2

Disponible a [enllaç](#).

Deformació de dos materials molt tous, amb deformació per compressió i expansió. Posterior ruptura per acoblament entre els cossos.

E	10^2
ν	0.4
hardening	10
t_c	0.25
t_s	0.15
Δt	10^{-5}
partícules	80776
model	corrotacional

B.34: hard5

Disponible a [enllaç](#).

Impacte de dos materials deformables i trencadissos contra una superfície, amb enduriment.

E	10^3
ν	0.3
hardening ₁	10
hardening ₂	0
Δt	10^{-5}
partícules	80776
model	corrotacional

B.35: hard6

Disponible a [enllaç](#).

Idèntic a l'anterior, però *Neo-Hookean*.

E	10^3
ν	0.3
hardening ₁	10
hardening ₂	0
Δt	10^{-5}
partícules	80776
model	<i>Neo-Hookean</i>

B.36: gelatin

Disponible a [enllaç](#).

Cub omplert de manera aleatòria, totalment elàstic. Rebot contra paret, i propagació de l'energia.

E	10^2
ν	0.45
plasticitat	desactivada
Δt	10^{-5}
partícules	100000
model	<i>Neo-Hookean</i>

B.37: C1

Disponible a [enllaç](#).

Estructura 'C', tota del mateix material tou. Deformació completa del material degut a la gravetat.

E	10^2
ν	0.45
plasticitat	desactivada
Δt	10^{-5}
partícules	105316
model	<i>Neo-Hookean</i>

B.38: C2

Disponible a [enllaç](#).

Idèntic a l'anterior però usant el model corrotacional.

E	10^2
ν	0.45
plasticitat	desactivada
Δt	10^{-5}
partícules	105316
model	corrotacional

B.39: C3

Disponible a [enllaç](#).

Estructura 'C', extrem de material molt tou i flàccid. Observar propagació de la deformació.

E_1	10^2
E_2	10^5
ν_1	0.45
ν_1	0.1
plasticitat	desactivada
Δt	10^{-5}
partícules	105316
model	corrotacional

B.40: C4

Disponible a [enllaç](#).

Idèntic a l'anterior, però *Neo-Hookean*.

E_1	10^2
E_2	10^5
ν_1	0.45
ν_1	0.1
plasticitat	desactivada
Δt	10^{-5}
partícules	105316
model	<i>Neo-Hookean</i>

B.41: C5

Disponible a [enllaç](#).

Estructura 'C', amb l'extrem lleugerament rígid, per evitar trencament, però deformable. Moviment de vaivé.

E_1	10^3
E_2	10^5
ν_1	0.45
ν_1	0.1
plasticitat	desactivada
Δt	10^{-5}
partícules	105316
model	<i>Neo-Hookean</i>

B.42: C6

Disponible a [enllaç](#).

Idèntic a l'anterior però usant el model corrotacional.

E_1	10^3
E_2	10^5
ν_1	0.45
ν_1	0.1
plasticitat	desactivada
Δt	10^{-5}
partícules	105316
model	corrotacional

B.43: C7

Disponible a [enllaç](#).

Estructura 'C', amb l'extrem lleugerament rígid. Trencament degut a una ν baixa, que evita la distribució de material davant la deformació.

E_1	10^3
E_2	10^5
ν	0.1
plasticitat	desactivada
Δt	10^{-5}
partícules	105316
model	corrotacional

B.44: C8

Disponible a [enllaç](#).

Idèntic a l'anterior, però *Neo-Hookean*.

E_1	10^3
E_2	10^5
ν	0.1
plasticitat	desactivada
Δt	10^{-5}
partícules	105316
model	<i>Neo-Hookean</i>

B.45: C9

Disponible a [enllaç](#).

Estructura 'C', amb l'extrem lleugerament rígid. Sense trencament.

E_1	10^3
E_2	10^5
ν_1	0.3
ν_2	0.1
plasticitat	desactivada
Δt	10^{-5}
partícules	105316
model	corrotacional

B.46: C10

Disponible a [enllaç](#).

Idèntic a l'anterior, però *Neo-Hookean*.

E_1	10^3
E_2	10^5
ν_1	0.3
ν_2	0.1
plasticitat	desactivada
Δt	10^{-5}
partícules	105316
model	<i>Neo-Hookean</i>

B.47: superDeff

Disponible a [enllaç](#).

Deformació elàstica portada a l'extrem. Comportament abstracte i rar, amb trencaments i forats. Pèrdua de la forma inicial a causa de l'acoblament. *Neo-Hookean*.

E	1
ν	0.49
plasticitat	desactivada
Δt	10^{-5}
partícules	105316
model	<i>Neo-Hookean</i>

B.48: superDeffCor

Disponible a [enllaç](#).

Idèntic a l'anterior, però corrotacional. Aquest manté molt millor el potencial energètic del cos, i per tant les forces de retorn.

E	1
ν	0.49
plasticitat	desactivada
Δt	10^{-5}
partícules	105316
model	corrotacional

C. Extractes de codi

En aquest apèndix s'hi troben diferents extractes del codi dut a terme durant el projecte.

El mateix codi és pot trobar en el documents adjunts a aquesta memòria.

C.1. Generar icosaedre

Mètode que crea un icosaedre des de zero. Usat per renderitzar les partícules. Fragment de codi extret de l'arxiu `SimVisualizer.cpp`.

```
1 // Generate the vertices and normals of an icosphere
2 // centered at the origin, with top/bottom-most vertices
   ↳ (0,+1,0)
3 // Returns: [x,y,z,nx,ny,nz] x 3 per face
4 // The normals are averaged for smoth interpolation in shading
5 std::vector<GLfloat> genIcosphere() {
6     constexpr float PI = 3.14159265358979323846f;
7     constexpr float H_ANGLE = (PI / 180) * 72; // 72 degree =
   ↳ 360 / 5
8     float V_ANGLE = std::atan(1.0f / 2.0f); // elevation =
   ↳ 26.565 degree
9
10    std::vector<glm::vec3> vertices(12);
11    float z = std::sin(V_ANGLE), xy = std::cos(V_ANGLE); //
   ↳ elevation
12    float hAngle1 = -PI / 2 - H_ANGLE / 2; // start at the
   ↳ second row. -126 degrees
13    float hAngle2 = -PI / 2; // -90 degrees. third row
14
15    vertices[0] = glm::vec3(0, 0, 1); // topmost vertex of
   ↳ radius 1
16
17    // all the vertices (10) between the top and the bottom
18    for (int i = 1; i <= 5; ++i)
19    {
20        int j = i + 5; // the simetric index
21
22        vertices[i] = glm::vec3(
23            xy * std::cos(hAngle1),
24            xy * std::sin(hAngle1),
25            z
26        );
27
28        vertices[j] = glm::vec3(
29            xy * std::cos(hAngle2),
```

```
30         xy * std::sin(hAngle2),
31         -z
32     );
33     hAngle1 += H_ANGLE;
34     hAngle2 += H_ANGLE;
35 }
36
37 vertices[11] = glm::vec3(0, 0, -1);
38
39 std::vector<int> indices;
40 indices.reserve(20 * 3);
41 auto addIndices = [&indices](int a, int b, int c)
42 {
43     indices.push_back(a);
44     indices.push_back(b);
45     indices.push_back(c);
46 };
47
48 int v0, v1, v2, v3, v4, v11;
49 v0 = 0;
50 v11 = 11;
51 for (int i = 1; i <= 5; ++i)
52 {
53     // 4 vertices in the 2nd row
54     v1 = i;
55     v3 = i + 5;
56     if (i < 5) {
57         v2 = i + 1;
58         v4 = i + 6;
59     }
60     else
61     {
62         v2 = 1;
63         v4 = 6;
64     }
65
66     // 4 new triangles per iteration
67     addIndices(v0, v1, v2);
68     addIndices(v1, v3, v2);
69     addIndices(v2, v3, v4);
70     addIndices(v3, v11, v4);
71 }
72
73 std::vector<glm::vec3> normals(indices.size());
74 for (int i = 0; i < indices.size(); i += 3)
75 {
76     glm::vec3 ab = vertices[indices[i + 1]] -
77         ↪ vertices[indices[i]],
78     ac = vertices[indices[i + 2]] - vertices[indices[i]];
```

```
78     glm::vec3 normal = glm::normalize(glm::cross(ab, ac));
79     if (glm::dot(normal, vertices[indices[i + 1]]) < 0)
80     {
81         // invert vertices and normal
82         normal = -normal;
83         std::swap(indices[i + 1], indices[i + 2]);
84     }
85
86
87     normals[i] = normal;
88     normals[i+1] = normal;
89     normals[i+2] = normal;
90 }
91
92 std::vector<glm::vec3> normals2(vertices.size(),
93     ↪ glm::vec3(0));
94 // smoth normals
95 for (int i = 0; i < indices.size(); ++i)
96 {
97     normals2[indices[i]] += normals[i];
98 }
99 for (glm::vec3& v : normals2)
100 {
101     v = glm::normalize(v);
102 }
103
104 std::vector<GLfloat> res(2 * 3 * indices.size());
105 for (int i = 0; i < indices.size(); ++i)
106 {
107     int p = 6 * i;
108     res[p + 0] = vertices[indices[i]].x;
109     res[p + 1] = vertices[indices[i]].y;
110     res[p + 2] = vertices[indices[i]].z;
111
112     res[p + 3] = normals2[indices[i]].x;
113     res[p + 4] = normals2[indices[i]].y;
114     res[p + 5] = normals2[indices[i]].z;
115 }
116 return res;
117 }
```

C.2. Parts del simulador

C.2.1. Matriu afí Q

Fragment de codi extret de l'arxiu Simulator_3D.cpp.

Computa de diferents maneres la matriu afí que incorpora el potencial elàstic del material.

```

1  // ----- AFFINE MATRIX ----- //
2  Eigen::Matrix3f affine;
3  if (this->mode == HYPERELASTICITY::COROTATED){
4      Eigen::JacobiSVD<Eigen::Matrix3f, Eigen::NoQRPreconditioner>
        ↪ svd(p.F, Eigen::ComputeFullU | Eigen::ComputeFullV);
5
6      const Eigen::Matrix3f r = svd.matrixU() *
        ↪ svd.matrixV().transpose();
7
8      //Corotated constitucional model
9      const Eigen::Matrix3f PF_t = (2.0f * mu * (p.F - r) *
        ↪ (p.F).transpose()) + (Eigen::Matrix3f::Identity() *
        ↪ (lambda * (J - 1.0f) * J));
10
11     const float Dinv = (4.0f * grid_size * grid_size);
12
13     const Eigen::Matrix3f stress = (-dt * p_prop.volume * Dinv)
        ↪ * PF_t;
14
15     affine = stress + p_prop.mass * p.C;
16 }
17 else if (this->mode == HYPERELASTICITY::NEOHOOKEAN){
18     // Neo-hookean times F~t
19     const Eigen::Matrix3f PF_t = (mu * ((p.F *
        ↪ (p.F).transpose()) - Eigen::Matrix3f::Identity())) +
20     (Eigen::Matrix3f::Identity() * (lambda * std::log(J)));
21     const float Dinv = (4.0f * grid_size * grid_size);
22
23     const Eigen::Matrix3f stress = (-dt * p_prop.volume * Dinv)
        ↪ * PF_t;
24
25     affine = stress + p_prop.mass * p.C;
26 }
27 else{
28     // SAND: No energy
29     affine = p_prop.mass * p.C;
30 }

```

C.2.2. Processament de la graella

Fragment de codi extret de l'arxiu `Simulator_3D.cpp`.

Processament de la graella, processant concretament els moments lineals allà emmagatzemats, aplicant físiques i forces externes.

```
1  // ***** GRID PROCESSING ***** //
2  for (unsigned int i = 0; i < grid_size; ++i){
3      for (unsigned int j = 0; j < grid_size; ++j){
4          for (unsigned int k = 0; k < grid_size; k++){
5              int idx = getInd(i, j, k);
6              Eigen::Array4f& cell = grid[idx]; // reference
7
8              if (cell.w() > 0){
9                  // ----- MOMENTUM 2 VELOCITY----- //
10                 cell /= cell.w();
11                 // Gravity
12                 cell.head<3>() += dt * g;
13
14                 // ----- LIMITS ----- //
15                 if (i < 2 && cell.x() < 0.0f){
16                     cell.x() = 0.0f;
17                 }
18                 else if (i > grid_size - 3 && cell.x() > 0.0f){
19                     cell.x() = 0.0f;
20                 }
21
22
23                 if (j < 2 && cell.y() < 0.0f){
24                     cell.y() = 0.0f;
25                 }
26                 else if (j > grid_size - 3 && cell.y() > 0.0f){
27                     cell.y() = 0.0f;
28                 }
29
30                 if (k < 2 && cell.z() < 0.0f){
31                     cell.z() = 0.0f;
32                 }
33                 else if (k > grid_size - 3 && cell.z() > 0.0f){
34                     cell.z() = 0.0f;
35                 }
36
37                 // ----- PHYSICS ----- //
38                 const Eigen::Vector3f& normalPhysics =
39                     ↪ physicsGrid[idx];
40                 // velocity dot normal
41                 float dot = normalPhysics.dot(cell.head<3>().matrix());
42                 // If oposed
43                 if (dot < 0.0f){
44                     // Remove normal velocity
```

```

44         cell.head<3>() = cell.head<3>() - (dot *
        ↪ normalPhysics).array();
45     }
46 }
47 }
48 }
49 }

```

C.2.3. Transferència $G2P$

Fragment de codi extret de l'arxiu Simulator_3D.cpp.
Transferència de velocitat de la graella a les respectives partícules.

```

1  int index; float w;
2  #pragma GCC unroll 3
3  for (int i = -1; i < 2; ++i){
4  #pragma GCC unroll 3
5      for (int j = -1; j < 2; ++j){
6  #pragma GCC unroll 3
7          for (int k = -1; k < 2; ++k){
8              w = weights[i + 1].x() * weights[j + 1].y() * weights[k
              ↪ + 1].z();
9              index = getInd(cell_i.x() + i, cell_i.y() + j,
              ↪ cell_i.z() + k);
10             grid[index] += w * moment_mass0;
11
12             moment_mass0.head<3>() += kstep;
13         }
14         moment_mass0.head<3>() += jstep;
15     }
16     moment_mass0.head<3>() += istep;
17 }

```

C.3. Producte exterior de tensors 3D

També anomenat *outer product*, equivalent a la multiplicació de dos vectors, amb el segon transposat, $\vec{a} \otimes \vec{b} = \vec{a} \cdot \vec{b}^T$. En Aquest mètode, el tensor es suma a un altre.

```

1  void Simulator_3D::SumOuterProduct(Eigen::Matrix3f& r, const
    ↪ Eigen::Array3f& a, const Eigen::Array3f& b)
2  {
3      // Totally unrolled outer product
4      r(0, 0) += a[0] * b[0];
5      r(0, 1) += a[0] * b[1];

```

```
6      r(0, 2) += a[0] * b[2];
7
8      r(1, 0) += a[1] * b[0];
9      r(1, 1) += a[1] * b[1];
10     r(1, 2) += a[1] * b[2];
11
12     r(2, 0) += a[2] * b[0];
13     r(2, 1) += a[2] * b[1];
14     r(2, 2) += a[2] * b[2];
15 }
```